Adabas D

	Version 13	User Manual UNIX

software AG

This document applies to Adabas D Version 13 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Table of Contents

User Manual Unix

This document covers the following topics:

Introduction

Connect

Adabas Tools: General Properties

Administration Tool Control

Loading Tool Load

End User Tool Query

Adabastclsh and Adabaswish

Programming Tool SQL-PL

C / C++ Precompiler

Cobol Precompiler

Call Interface (ODBC)

Call Interface (JDBC)

Call Interface (OCI)

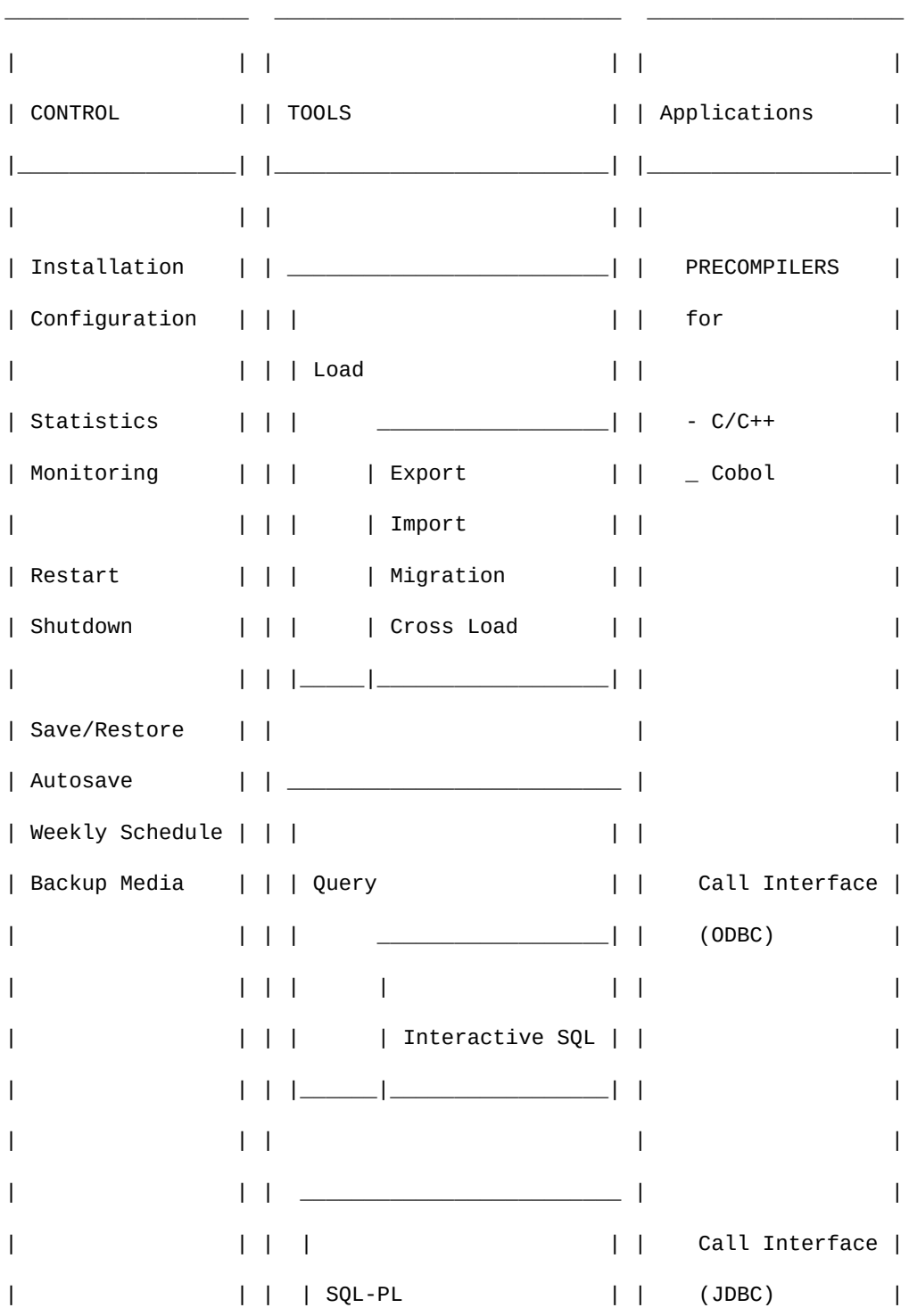
Call Interface (Perl)

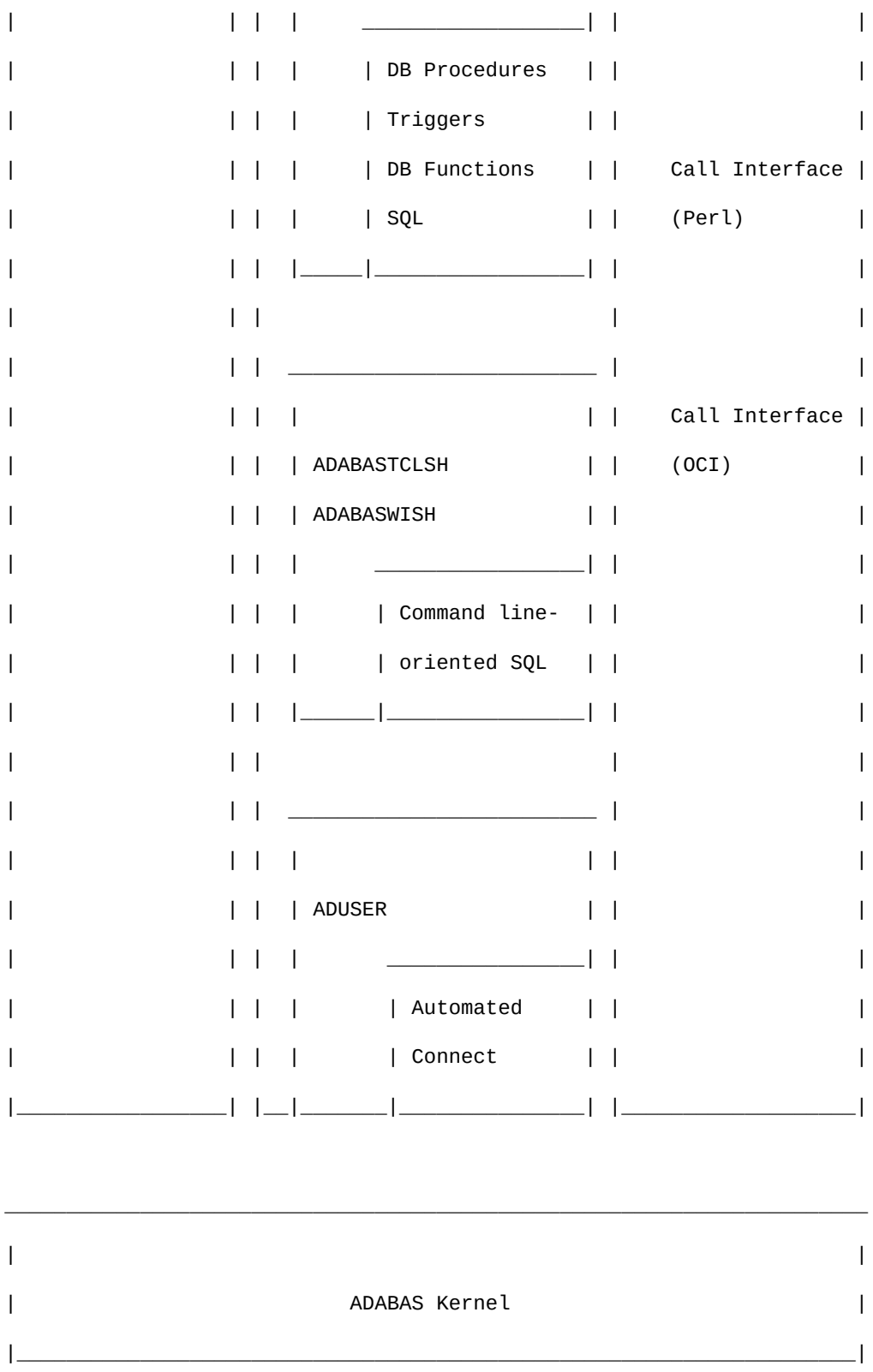
Appendix - Keyboard Layouts

Introduction

What Is the Purpose of This Manual?

The "User Manual Unix" describes operating system-specific aspects of the work with Adabas under Unix. It contains the information needed to call and use the Adabas tools and the Adabas programming interface under Unix. The following interfaces to the database kernel are available to the Unix user:





The Control and Query tools can be called in two variants: one with character-oriented interface on any alphanumeric terminal or one with the Tcl/TK interface which is also platform-independent. The GUI variant does not yet support the full functionality of the tools for the current Adabas version (for more details refer to the corresponding manuals).

What Are the Prerequisites ?

This manual does not enter into the particulars of the general usage of Adabas and its components. It is meant to be a complement of the individual Adabas manuals which contain a detailed explanation of the usage of the Adabas tools and programming interface as well as of their functionalities. It is assumed that the reader of this manual is already familiar with the main functionality of the Adabas components used by him.

Who Should Use This Manual?

This manual provides additional information for any user who works with Adabas under Unix including

- the database administrator or database operator who, as Control user, must perform administrative tasks or, as Load user, must load external data or edit own data for further external processing.
- the Adabas user who usually works with the end user tools or who accesses the database using precompiled application programs.
- the application programmer who uses the precompilers to construct and test Unix application programs with embedded SQL.

What Is the Structure of This Manual?

Each section of the "User Manual Unix" is directed to a certain kind of user.

First of all, the connect of a user to the database is described in Section "Connect". The connect procedure is the same for almost all the Adabas components. The user specifications required for the connect can be recorded by using the special tool ADUSER (see Sections "Connect With Predefined User Specifications (ADUSER)" and "Using ADUSER". The other Adabas components can retrieve them from there.

Section 3, explains operating system specific properties that are the same for all tools such as file handling, printing, editing etc., as well as working with the editor built into some of the tools (see Section, "The Built-in Editor for Load and Query").

Section 4 explains the call options of the "Administration Tool Control".

Section 5 describes the call and properties of the "Loading Tool Load".

Section 6 describes the "End User Tool Query" and its operating system specific functions and properties.

The following Section 7 describes "Adabastclsh and Adabaswish".

Section 8 describes the "Programming Tool SQL-PL" and its operating system specific functions and properties.

Sections 9 to 14, "C / C++ Precompiler", "Cobol Precompiler", "Call Interface (ODBC)", "Call Interface (JDBC)", "Call Interface (OCI)" and "Call Interface (Perl)" are primarily directed to the application programmer who uses one of the Adabas programming interfaces to construct application programs with embedded SQL.

The Appendix contains the keyboard layouts .

Connect

This chapter covers the following topics:

- Establishing a Database Session
- Using ADUSER

Establishing a Database Session

To establish a database session, the Adabas user must connect to the database. To be able to do so, certain user specifications must be passed to the called Adabas component for identification purposes.

The following information is required for the connect:

USERID:	Adabas user name
PASSWORD:	Adabas password of the user
SERVERDB:	name of the Adabas database to be used
SERVERNODE:	name of the network node where the addressed database is located

The SERVERNODE specification is not necessary when a SERVERDB of the local computer is used.

In addition to the required user specifications listed above, more optional user specifications such as TIMEOUT and ISOLATION LEVEL may be passed to the Adabas tool when connecting (see Section "Using ADUSER").

To access the database from an Adabas tool or precompiler program, you may pass user specifications to the component in four ways:

1. **User Specifications Predefined With ADUSER**

   User specifications can be preset and stored by using the special tool ADUSER. The ADUSER entries can be accessed when an Adabas tool (except CONTROL) or a precompiler program is called.

2. **User Specifications Made When Calling a Tool**

   User specifications can be passed as arguments. For example:

   xquery -u parker,secret -d testdb -n sql1

3. **Predefining by Using Environment Variables**

   The database name can be predefined by setting the environment variable SERVERDB.

4. **Connect Screen**

If there are no predefined user specifications, the called Adabas tools returns the connect screen. This does not happen when precompilers and application programs are called.

The connect screen is also displayed when the user specifications did not result in a successful connect.

Missing or incorrect user specifications in a precompiler program have the effect that a program is aborted and a corresponding error message is output.

User specifications may be passed to the Adabas tools using a combination of these four ways. The combinations are described in Section "Precedence Rules of the Various Connect Procedures".

If precompilers and application programs constructed using the Adabas programming interface are called, user specifications can be passed directly within the program. Different precedence rules apply in these cases; they are described in the "C/C++ Precompiler" or "Cobol Precompiler" manual. In principle, the same ways of connecting are valid for both precompilers and application programs. These principles are explained in the following for calls to the Adabas tools.

This section covers the following topics:

- Connect With Predefined User Specifications (ADUSER)
- Connect With User Specifications When Calling a Tool
- Precedence Rules of the Various Connect Procedures

Connect With Predefined User Specifications (ADUSER)

The simplest way to call an Adabas tool is to use predefined user specifications. The specifications must have been predefined using the ADUSER tool.

For one operating system user, ADUSER manages up to 32 different combinations of user specifications for establishing an Adabas session. These specifications are stored in the file .ADUSER (in the following also referred to as ADUSER file) in the user’s HOME directory.

In this way, user specifications can be predefined for different tasks and then be used for the connect. Thus it is possible to administer individual user specifications even for several database users who work under different Adabas user names but in the same HOME directory.

The user receives his specifications from the database administrator who must have created the corresponding database user. The user himself may store these specifications in the ADUSER file by calling "ADUSER". (Section "Using ADUSER" contains a detailed description of ADUSER.)

When valid predefined user specifications are used to call an Adabas tool, the operative mode of the tool can be accessed automatically.

The syntax of the connect with ADUSER access is in general:

```
<component name> [-U <user option>]

    <component name> ::=  adquery  | xquery | xload

    <user option>    ::=  <userkey> | prompt
```

The Sections "C / C ++ Precompiler", "Cobol Precompiler", and "Call Interface (OCI)" of this manual and the "C/C++ Precompiler" or "Cobol Precompiler" manual describe how the predefined user specifications are used for precompilers and precompiler programs as well as for the OCI. Applications using the ODBC Interface do not access the ADUSER data.

This section covers the following topics:

- Calling Without Parameter Specifications
- Calling With USERKEY
- Calling With "prompt" Option

Calling Without Parameter Specifications

This will be the most common format of the call.

When using this call for the end user tools, all user specifications required for the connect are taken from the parameter combination "DEFAULT" which must have been stored using ADUSER (see Section "Using ADUSER"). After the call, the Adabas tool is operative.

Calling With USERKEY

To use one of the other parameter combinations stored with ADUSER for a connect, this parameter combination must be addressed with the option -U and its key name (USERKEY). The USERKEY must be specified exactly as it is defined in the ADUSER file; i.e., the USERKEY is case sensitive.

Example:

Besides the usual user specifications in the parameter combination "DEFAULT" declared using ADUSER, the user frequently works with another parameter combination, e.g., to access a database on another computer. The user specifications required for this purpose have been stored in ADUSER with the key "remsql". The call then runs as follows:

xquery -U remsql

The user specifications are taken from the parameter combination "remsql"; the Adabas tool is accessed automatically.

Calling With "prompt" Option

If the connect screen is to be displayed in any case, this can be obtained by using the "-U prompt" option:

xquery -U prompt

In this case, the user specifications are preset from the ADUSER parameter combination "DEFAULT" and the connect screen displayed so that the user can overwrite the specifications, if necessary.

Connect With User Specifications When Calling a Tool

The user specifications are passed as arguments with the call of the Adabas tool.

The syntax of the call is in general:

```
<component name> <connect spec>


    <component name> ::=  adcontrol | xcontrol | xload  adquery | xquery

    <connect spec>   ::=  [-u <user id>[,<password>]]

                          [-d <serverdb>] [-n <servernode]

                          [-t <session timeout>]

                          [-I <isolation level>]
```

The options -t, -I, and -n cannot be used for xcontrol.

Example:

xquery -u parker,secret -d testdb -n sql1 -t 300

All user specifications are made explicitly, and no more information from the ADUSER file is required. The tool is accessed automatically. If the specifications for the options -u, -d, or -n are incorrect, the connect screen displayed where the entries can be corrected.

Precedence Rules of the Various Connect Procedures

This section only refers to the Adabas tools. For the call of precompilers and application programs, there are special precedence rules of passing the user specifications. These rules are described in the "C/C++ Precompiler" or "Cobol Precompiler" manual.

When calling an Adabas tool, the following order of precedence applies (highest priority first):

Connect data is passed with parameters when calling the tool,

Connect data is taken from the ADUSER file,

SERVERDB is taken from the Unix environment variable SERVERDB

i.e., each procedure of higher priority overrides the specifications of a less-priority procedure.

In detail, the following is true:

1. If the corresponding parameters for the required user specifications USERID, PASSWORD, SERVERDB, and SERVERNODE have been set for the call of an Adabas tool, these parameters are used to establish a database session. The operative mode of the tool can be accessed automatically. Example:

   xquery -u parker,secret -d dbtest -n sql1

   The same is true if the called tool uses additional user specifications such as TIMEOUT or ISOLATION LEVEL. DEFAULT values possibly existing from an ADUSER file or from Unix environment variables are overridden.

2. If ADUSER specifications are available, all missing and required user specifications are taken from the parameter combination "DEFAULT".

   Examples:

   xquery -u parker,secret

   xquery -u parker -d dbprod -n sql1

   xquery -u parker -d testdb

   xquery -u parker

   xquery -d testdb

   In all cases, at least one of the required user specifications is missing:

   - SERVERDB and SERVERNODE or
   - PASSWORD or
   - PASSWORD and SERVERNODE or
   - PASSWORD, SERVERDB, and SERVERNODE or
   - USERID, PASSWORD, and SERVERNODE.

   The missing specifications are taken from the ADUSER parameter combination "DEFAULT". The operative mode can be accessed automatically.

   xquery

   All user specifications are taken from the ADUSER parameter combination "DEFAULT". The operative mode can be accessed automatically.

3. If a special USERKEY was specified before the explicit specification of individual user parameters, the missing user specifications are completed from the corresponding ADUSER parameter combination.

   Examples:

   xquery -U special -d dbprod

   USERNAME, PASSWORD, and SERVERNODE are completed from the ADUSER parameter combination "special".

   xquery -U remsql

   All user specifications are taken from the ADUSER parameter combination "remsql". The operative mode can be accessed automatically.

4. If only the parameter SERVERDB is missing and no ADUSER specifications exist, then the value of the environment variable SERVERDB is used to complete the user specifications.

5. If one of the required specifications cannot be found in any of these sources or if one of the specifications is incorrect, the called Adabas tool returns the connect screen.

Using ADUSER

This section covers the following topics:

- Calling ADUSER
- Structure of the ADUSER Input Form
- Creating an ADUSER File in Batch Mode

Calling ADUSER

Format:

aduser [-u <user id>[,<passwd>]] [-b <filename>]

ADUSER distinguishes between the first and subsequent calls. It is not possible to specify options for the first call. For the first group of parameters, the input screen displayed at once. For all the other calls, it is necessary to connect with USERID and PASSWORD from the first parameter combination that contains a non-empty USERID.

The connect can be done with the option -u in the call or from the connect screen of ADUSER.

The option -b allows ADUSER to be used in batch mode (see Section 2.2.3, "Creating an ADUSER File in Batch Mode").

Structure of the ADUSER Input Form

Up to 32 parameter combinations can be stored. Each consists of

User Key

Key name used to access the combination.

The first parameter combination is named "DEFAULT". This name cannot be modified.

User Name

Adabas user name.

Password

Adabas password of the user.

Server DB

Name of the Adabas database to be used. If not specified, the name will be taken from the environment variable SERVERDB.

Server Node

Name of the network node where the addressed database resides. If not specified, the local computer will be taken.

SQL Mode

Ensures compatibility with the SQL dialects of other manufacturers. Possible specifications are ADABAS, ANSI or ORACLE. Default is ADABAS. This parameter is effective for precompiler programs and the tools Load and Query.

Cachelimit

Limit for the size of a temporary data buffer (only affects application programs with large SELECT results).

Timeout

Time interval in seconds at the end of which an inactive session of the user is terminated: see the "Reference" manual, Section "Transactions, <connect statement>".

Isolation Level

ISOLATION LEVEL for locks that affect the user (only valid for application programs, precompilers): see the "Reference" manual, Section "Transactions, <connect statement>".

The password is invisible and must be entered twice for security reasons.

User Key, Server DB and Server Node are case sensitive.

User Name and Password must be enclosed in double quotation marks, as in database operation, if they are to contain lowercase letters or special characters. Otherwise, lowercase characters are converted into uppercase.

The SQL Mode can be specified in any notation. If not specified, the default value ADABAS is valid.

Cachelimit, Timeout and Isolation Level are numeric parameters. If the respective default value is to be used for these parameters, -1 must be specified as value. In the empty input screen, the default values are already set for these parameters.

The current number of the group of parameters is displayed in the header line of the screen. One group is displayed per page.

The following functions can be executed by using the available buttons:

Cancel

Leaving ADUSER without saving. Modifications previously stored with Save are rolled back.

Clear

Removing the entries of the current combination.

Delete

Deleting an individual combination. Subsequent parameter combinations move upward. Note: The parameter combination moved to the first place is automatically assigned the User Key "DEFAULT". The deletion only becomes effective if ADUSER is left with Save.

Delete All

Deleting all combinations.

Ok

Leaving ADUSER.

Save

Saving the current parameter combinations.

Creating an ADUSER File in Batch Mode

The ADUSER file cannot only be created in interactive mode by entering the parameters in the input forms but also using a batch file which must be specified with the ADUSER call.

The call for the batch mode is:

aduser -b <filename>

The name of the file can be chosen freely. The file consists of groups of nine lines. The first line of each group contains the User Key, the second the User Name, the third the Password, then follow Server DB, Server Node, SQL Mode, Cachelimit, Timeout and Isolation Level just as they are specified in the input screen. The next group (parameter combination) begins in the next line. If optional parameters (Server DB, Server Node, SQL Mode, ...) are not to be entered, a blank line must be at the corresponding place. The entries in the file begin in column one without field identifier, for example:

DEFAULT

parker

secret

db1dial

sqldial

ADABAS

-1

-1

home

parker

"top_secret"

db2dial

sqldial

90

1

-1

When the option -b is used, a new ADUSER file is created in any case. An ADUSER file possibly existing will be overridden.

If the specified file has the length 0, the state after the installation is restored; i.e., the input screen for the first parameter group appears with the next ADUSER call.

Both formats can be used to make ADUSER operative again when the user forgot the password.

Adabas Tools: General Properties

The following two Sections "Special Call Options" and "Case Sensitivity of Database Objects" are valid for all Adabas tools in both variants. The other sections of Section 3 only refer to the character-oriented variants xload and xquery, not to the Tcl/Tk-based GUI tools.

This chapter covers the following topics:

- Special Call Options
- Case Sensitivity of Database Objects
- Using Files
- Printing from the Adabas Tools
- Calling Operating System Commands
- The Built-in Editor for Load and Query
- The System Editor

Special Call Options

All Adabas tools can be called from the shell or a shell script.

They support the following special call options:

-h

displays the call options possible for the respective tool.

-V

displays the version of the tool.

A database session is not opened.

Format:

```
  <component name> -h
| <component name> -V


  <component name> ::= adcontrol | xcontrol | xload | adquery | xquery
```

Example:

xquery -h

Result:

```
correct use of xquery is:


connect user        ::= -u <userid>,<password>
database            ::= -d <serverdb>
nodename            ::= -n <servernode>
ADUSER key          ::= -U <userkey>
timeout             ::= -t <sec>
version information ::= -V
help information    ::= -h
run file            ::= -r <filename>
batch file          ::= -b <filename>
run query object    ::= -R [<owner.>]<name>
batch query object  ::= -B [<owner.>]<name>
export object       ::= -e <object_name>,<filename>
import object       ::= -i <filename>
list mode           ::= -L
select mode         ::= -s
SQL mode            ::= -S  ADABAS
append              ::= -A
```

Example:

adquery -V

Result:

QUERY Version 12 Date 2002-01-31

Case Sensitivity of Database Objects

As a general rule, the Adabas tools convert all input characters from lowercase letters into uppercase. As a consequence, database objects are usually stored and then accessed with uppercase names, regardless of the format used on input.

It is possible to bypass this conversion and explicitly give the database objects lowercase names by enclosing the names in double quotation marks when creating or calling a database object (see the "Reference" manual, Section "Common Element, <token> - <special identifier>" and the manuals of the Adabas tools).

If the names of stored database objects containing lowercase characters are to be passed as parameters when an Adabas tool is called from the shell, these names must be enclosed in double quotation marks. They must also be masked so that they will not be removed from the shell.

Examples:

The call

xquery -R hotel

has the effect that Query exec command HOTEL that was stored in Query either with the command ==> store HOTEL or ==> store hotel. The call of the lowercase command hotel stored with ==> store "hotel" must be formatted as follows:

xquery -R ’"hotel"’

These examples can be applied when calling all other Adabas tools.

Using Files

Some commands within the Adabas tools have a filename as argument. This filename always refers to a file in the Unix file system, i.e., to a regular file or a device (e.g., a tape device).

Examples:

put customer.data

get parker/customer.data

export customer /dev/rmt0 -32

1. Files in the File System:

   The filename must comply with the Unix conventions. Upper- and lowercases must therefore be distinguished. They will not be converted.

   Examples of Unix filenames:

   1. 1. customer.form
   2. 2. parker/customer.form
   3. 3. /u/parker/customer.form
   4. 4. ${HOME}/customer.form

5. 5. /dev/rmt0

The Unix filename is specified either as a simple filename (Example 1), or as a relative path name, i.e., starting with one or more directory names that are separated from each other and from the simple filename by a "/" (Example 2), or as an absolute path name starting with the root directory "/" (Example 3).

For simple filenames, the current working directory will be scanned. For relative path names, the specified directories will be searched, starting with the current working directory.

Each simple filename or directory name not contain a "/". The total length of a path name must not exceed 64 characters.

Environment variables of the Unix environment can be used within filenames (Example 4). When doing so, the variable names must be enclosed in curly braces.

Successful file accesses require that the user has the necessary Unix privileges for reading and writing files or directories.

Files with variable record lengths to be read by the Adabas tools must contain end-of-line characters. Files with fixed record lengths must contain the corresponding length as a 4 byte integer in the first record of the file.

2. Tape Files:

   When reading from or writing to tape out of Load or Query, a blocking factor of 4, 16, or 32 KBytes can be specified if the tape device permits such a specification. The default value is 4 KBytes. Larger values must be passed as parameters (see Example 3 at the beginning of this section). For the SAVE/RESTORE functions of Control, a blocking factor of 32 Kbytes must always be used.

3. STDIO

   The Unix-specific standard input/output can also be used, e.g.;

   - when "STDOUT" is used as the output filename:

     Using the command

     ```
     select * from test

     REPORT

     put STDOUT
     ```

     stored in Query with the name "OUT", e.g., the result of a request can be written directly to the file "outfile":

     ```
      xquery ... -B out > outfile
     ```

   - within a shell script:

```
xload -b STDIN << +

create table test

 ...

+
```

- using a pipe served from a program or by running a file

```
cat file | xload -b STDIN
```

Printing from the Adabas Tools

Unless another specification was made in the Print Format menu of the Adabas tools, output generated by the PRINT command or the function key "PRINT" is directed to standard lp.

In Load and Query the Print Format menu can be called from the Set parameters screen; in Control, it can be called using the "Configuration / Alter Parameters / Set Defaults" menu item. If output is to be directed to another printer, the Unix device name of the desired printer must be specified instead of "lp" for "printer" in the Print Format menu. When printing with the corresponding print format, the command

lp -d <printer device name>

is used instead of "lp" (also see the corresponding sections in the manuals of the Adabas tools).

As the whole content of the "Printer" input field is passed (up to the maximum length of 64 characters), it is possible to specify here any other parameter supported by the selected printer driver. Example: The specification

md07 -H resume -n 5

has the effect that the data to be printed is processed with the call

lp -d md07 -H resume -n 5

To output the result of a DATAEXTRACT run or the protocol file to the printer in Load, the filename "PRINTER" (note the uppercases!) must be specified. The target printer used in this case is also the printer specified in the currently valid print format.

Calling Operating System Commands

In all Adabas tools (except Control), Unix shell commands and executable programs can be called from the command line by prefixing an exclamation mark to them.

Examples:

```
====>  ! ls -l             Synchronous display of the current

                           directory; i.e., work in the Adabas

                           tool will be interrupted.
```

```
====>  ! lp out &           Asynchronous background print
                           output of the file "out".

====>  ! appl > appl.out &  The program appl is
                           (asynchronously) started
                           in background writing the
                           result to the file appl.out.
```

The commands are started in a shell of their own which terminates when control is returned to the calling Adabas tool. For example, it is therefore not possible to change the current directory of an Adabas tool by "!cd".

Syntax:

```
<unix command> ::= !<synchronous unix command>
                 | !<asynchronous unix command>&
                 |  exec <synchronous unix command>
                 |  exec async <asynchronous unix command>

<synchronous unix command>  ::= any Unix command or
                                any program call

<asynchronous unix command> ::= any Unix command or
                                program which neither
                                uses stderr nor stdout
                                (because otherwise no correct
                                screen display can be
                                guaranteed)
```

The Built-in Editor for Load and Query

As various kinds of editors are used in different operating systems, Adabas provides two types of a built-in editor for the Adabas tools Load and Query. One type uses key functions which follow the pattern of the RAND editor (Unix); the other uses prefix commands similar to those of the XEDIT. You can switch interactively between these two variants.

Both editor types support additional editor commands that can be entered in the command line.

The Key-oriented Editor

This editor is called by the command RED.

It is key-oriented, i.e., most of the functions (insert, delete, move) can be called by using special keys.

The particular assignment of these special keys is included in Appendix.

Marking Text Areas

Before the functions can be executed, the corresponding text areas must be marked.

To execute functions that only refer to one line, position the cursor on the desired line and then press the function key.

To mark an area (e.g., several lines, rectangles), position the cursor at the beginning of the area and press the Mark key. Then position the cursor at the end of the area and press the desired function key.

If the cursor is only moved vertically, the total length of lines is marked.

If the cursor is also moved horizontally, a rectangle (block) is defined that is limited by these two marks.

Inserting Lines and Blocks

To insert single blank lines, position the cursor on the corresponding line and press the key INS-B.

To generate several blank lines from the cursor position, issue the command

CMD n INS-B

where n is the number of the desired blank lines.

If an area has been marked, a corresponding number of blank lines or a rectangle of blanks is inserted.

Deleting Lines and Blocks

To delete single lines, position the cursor on the corresponding line and press the key DEL-B.

To delete several lines from the cursor position, issue the command

CMD n DEL-B

where n is the number of lines to be deleted.

If an area has been marked, a corresponding number of lines or the rectangle is deleted.

The last deleted text is stored in a temporary buffer (PICK buffer).

To restore the last deleted text, press the Put key.

Copying Lines and Blocks

To copy the line on which the cursor is placed or the marked text area, write it to a temporary buffer (PICK buffer) by pressing the Pick key. Then copy this text to any place by pressing the Put key.

As the PICK buffer is preserved up to the next PICK command or until the DEL-B key is pressed, its contents can be copied as often as desired.

Moving Lines and Blocks

To move lines or marked text areas, delete them from one place by pressing the DEL-B key. Then restore them immediately afterwards to another place by pressing the Put key.

The Prefix Editor

The prefix-oriented editor version is called using the command XED.

The prefix commands are written to the area of the form marked by "====".

Prefixes may be positioned either at the left or at the right margin of the input area by using the SWITCH command. The SWITCH command must be entered in the command line.

The prefix commands refer to single lines or blocks of lines of the edit form. Default for the block length n is 1.

In

After this line, n new lines are inserted and initialized with blanks.

Dn

Starting from this line, n lines are deleted.

DD

Starting from this line, all lines are deleted up to the next line marked accordingly.

>n

The contents of this line are moved n columns to the right.

>>n

The contents of the lines from this one to the next one that is also marked with >> are moved n columns to the right.

<n

The contents of this line are moved n columns to the left.

<<n

The contents of the lines from this one to the next one that is also marked with << are moved n columns to the left.

"n

This line is duplicated n times.

""

The series of lines from this one up to and including the next one that is also marked with "" are duplicated.

""n

The series of lines from this one up to and including the next one that is also marked with "" is duplicated n times.

C

This line is copied after the next line marked with F or before the next line marked with P.

CC

The series of lines from this line up to and including the next one that is marked accordingly is copied after the line marked with F or before the line marked with P.

CCn

The block of lines from this line up to and including the next one that is marked with CC is copied n times after the line marked with F or before the line marked with P.

M

Like C, but deleting the original line.

MM

Like CC, but deleting the original lines.

Xn

The next n lines are excluded from the display.

XX

The series of lines from this line up to and including the next one that is similarly marked is excluded from the display.

Sn

The next n excluded lines are displayed.

S-n

The last n excluded lines are displayed.

/

The corresponding line is centered on the screen.

The commands (I, D, DD, C ...) can also be entered in lowercases.

General Commands

The editor commands are entered in the command line after ===>.

All keywords can be abbreviated to three characters. They can be written in upper- or lowercase characters.

Some commands can also be executed by using function keys. The current meaning of the function keys is displayed on the screen.

GET Command

1. The content of an external file is copied into the input area.

   Call:

   GET <file name> [<section>]

   <section> ::= <beginning> [<number>]

   The target position is the line in the input area on which the cursor is placed, or the first line of the form if the cursor is not positioned within the input area.

   The sequence number of the first line to be copied (default: 1) and the number of the lines to be copied (default: as many as possible) can be optionally specified. At most 12 KB can be copied in both cases. If the specified file exceeds this value, the rest will be truncated and a message be output.

   Examples:

   get clist.query

   get clist.form 20

   get clist 100 18

2. The content of the internal PICK buffer is copied into the input area.

   Call:

   GET

   Target position is the line in the input area on which the cursor is placed, or the first line of the form if the cursor is not positioned within the input area.

Example:

get

PUT Command

1. The content of the edit form is copied into a file.

   Call:

   PUT <file name> [<section>] [APPEND]

   <section> ::= <beginning> [number>]

   The number of the first line to be copied (default: 1) and the number of the lines to be copied (default: as many as possible) can be optionally specified.

   Specifying APPEND ensures that text is added at the end of an already existing file rather than overwriting the file.

   Examples:

   put clist.query

   put clist.form 20 append

   put clist 100 18 app

2. The content of the input area is copied into the internal PICK buffer.

   Call:

   PUT [<number>]

   The first line copied is the first line displayed on the screen. The user optionally specify the number of lines to be copied (default: the lines displayed on the screen).

   Examples:

   put

   put 3

PRINT Command

This command sends the contents of the edit form to the print log.

Call:

PRINT

PRINT writes the content of the form into the currently opened print log. The command can also be issued by pressing the Print key.

CLOSE Command

This command closes the print log and sends its content to the printer.

Call:

CLOSE

CLOSE terminates the currently opened print outputs the log to the printer.

When the tool that called the editor is left, a print log not yet printed is automatically send to the printer.

If a PRINT command was issued by using the Print key, the print log is sent to the printer by immediately pressing the key a second time.

SEARCH Command

This command searches a specified character string.

Call:

[-]/ <character string> /

Starting from the first displayed line, the first occurrence of the specified character string is searched. If it is detected, the corresponding line is highlighted on the screen and marked by the cursor.

REPLACE Command (CHANGE)

REPLACE or CHANGE replaces character strings in the edit form.

Call:

REPLACE / <char_string_old> / <char_string_new> / [<area>]

or

CHANGE / <char_string_old> / <char_string_new> / [<area>]

```
<area>   ::= <n> [<m>]

<n>        ::= <lines to change>

<m>       ::= <changes per column>
```

Default values for the area is n = 1, m = 1; i.e., starting from the first displayed line, each occurrence of <char_string_old> is replaced by <char_string_new>.

n indicates the number of lines in which replacements are to be performed.

m indicates the maximum number of replacements per line.

Specifying * * replaces any occurrence of <char_string_old> up to the end of the file.

The SPLTJOIN Key

The Spltjoin key splits and rejoins single lines of text.

A line is split or joined from cursor position. If the cursor is placed behind the end of a line, the text following the cursor will be appended to the current line.

Additional Commands

RESET

clears the input area.

UP <n>

scrolls towards the top of the form for n lines (n is optional).

- <n>

same function as UP.

DOWN <n>

scrolls towards the bottom of the form for n lines (n is optional).

+ <n>

same function as DOWN.

LEFT

moves the window towards the left margin of the form.

RIGHT

moves the window towards the right margin of the form.

TOP

moves the window to the top of the form.

BOTTOM

moves the window to the bottom of the form.

SPLIT

If the cursor placed in the input area is positioned to the command line by using the key CMD or Enter and the command SPLIT is entered, the line is split at the position where the cursor was placed.

JOIN

If the cursor placed in the input area is positioned to the command line by using the key CMD or Enter and the command JOIN is entered, the next line is appended to the line where the cursor was placed.

WRAP ON

The command is only available in the RED and only if the terminal used is appropriate for it. If there are lines in the editor area that are longer than the editor window, the cursor is positioned to these lines and an error message is displayed.

WRAP OFF

disables the automatic split/join function.

WRAP

shows whether the automatic split/join function is enabled or disabled.

=

writes the last executed editor command to the command line.

==

repeats the last executed editor command.

?

calls the HELP function of the editor.

The System Editor

Under Unix, all Adabas tools can also call a system editor. Default is the Unix editor "vi". Any other editor installed on the system can be called when it supports the same call syntax as the "vi". In Load and Query, the desired editor is specified in the Set parameters screen; in Control, it is specified using the "Configuration / Alter Parameters / Set Defaults" menu item.

While the call of a system editor only facilitates the reading of protocol files in Control, the defined system editor can be used instead of the built-in editor in Load and Query.

Entering the command "sysedit" in the command line of the built-in editor switches to the selected system editor, passing the contents of the edit form to it. Modifications to the contents must be saved within the system editor, if they are to be kept when returning to the Adabas tool.

To transfer the editor contents, a file is generated in the current directory. This file will be deleted after returning to the built-in editor.

The name ed."pid" is used for this file. "pid" denotes the string obtained from the process ID.

In Control, a copy of the corresponding protocol file is created in the directory $DBROOT/wrk/$SERVERDB/cn_tmp, loaded into the editor, and deleted after leaving the editor.

Administration Tool Control

Control is available in two variants. The selection is done with the call command.

adcontrol

calls a GUI interface based on Tcl/TK. This interface does not yet provide the complete functionality of Control (compare the "Control" manual).

xcontrol

displays a character-oriented interface for an alphanumeric terminal.

This chapter covers the following topics:

- Prerequisites on Operating System Level
- Calling Control
- Backup Files

Prerequisites on Operating System Level

For regular operation, the Unix user who calls Control must satisfy some conditions.

To be able to call Control, the user must be authorized to write into the directory that during installation was denoted as RUNDIRECTORY, because the Control protocol files are stored there (see Section "Backup Files").

All SAVE and RESTORE functions can be executed on tape devices, regular files, or using pipes. In general, the rules specified in Section "Using Files" apply.

Writing to tapes and into pipes is always done in blocks of 32 KB.

To be able to install automatic backup operations using the Backup/Schedule Manager (see the "Control" manual), the Unix user who calls Control must be authorized to create crontab files.

To ensure a correct administration of such automatic backup operations, Control must always be called by the same Unix user.

Calling Control

Format:

Call:	Call:
xcontrol [<connect spec>]	adcontrol [<connect spec>]
[<batch operation spec>]	[<tcl commands>]
| xcontrol -V	| adcontrol -V
| xcontrol -h	| adcontrol -h
Call options:	Call options:
<connect spec> ::=	<connect spec> ::=
[-u <userid>[,<password>]]	[-u <userid>[,<password>]]
[-d <serverdb>]	[-d <serverdb>]

Control distinguishes between the first call after an installation and the calls following thereafter.

For the first call, no parameters must be specified. Control displays an input form for defining a profile that allows for storing all user names and passwords required for the administration of the database. If the usernames and passwords have been entered, the main screen of Control is displayed.

For further calls without options, a connect screen displayed which contains the options required for the connect. The options -u and -d are described in Section "Connect", the options -V and -h in Section "Adabas Tools: General Properties".

The utilities "xbackup" and "xrestore" provide batch functions of the SAVE and RESTORE operations in Control. Their functioning and the complete call syntax under Unix are described in the "Control" manual.

Backup Files

All SAVE and RESTORE functions can be performed on tape devices, regular files, or using pipes. In general, the rules specified in Section "Using Files" apply.

Writing to tapes and into pipes is always done in blocks of 32 KB.

Loading Tool Load

This chapter covers the following topics:

- Calling Load
- Load Protocol File
- Load Return Codes

Calling Load

Format:

```
Call:


  xload [<connect spec>] [<commandfile spec>]
| xload [<connect spec>] [<LOAD command>]


| xload -V
| xload -h



Call Options:


<connect spec>  ::=     [-U <user option> ]
                   [-u <user id>[,<password>]]
                   [-d <serverdb>] [-n <servernode]
                   [-t <session timeout>]
                   [-S ADABAS]


<commandfile spec>      ::=     -r <filename> [-P ] [<parameter list>]
                     | -b <filename> [<parameter list>]



Parameters:
```

```
<user option>   ::=     <userkey> | prompt


<parameter list>        ::=     <parameter> [<blank> <parameter list>]
```

"xdbload" can be used instead of "xload". This is recommended for Unix systems that provide a user command "xload".

Calling Load (general format)

xload or xdbload

The options -u, -U, -d, and -n required for the connect are described in Section "Connect", the options -V and -h in Section "Adabas Tools: General Properties".

After the connect, the tool is in input mode where the Load commands can be entered.

Specifying a TIMEOUT Value

The SESSION TIMEOUT value determines the time interval in seconds at the end of which the session will be terminated if it was not active. The database administrator can determine this value for the whole database using CONTROL or for a single user on his creation (default: 300 seconds). The option -t allows the user to specify a smaller value in seconds. A value larger than predefined produces an error message.

xload -t 90

The database session started with this call is terminated after 90 seconds of inactivity.

Specifying an SQLMODE

The option -S can be used to specify the SQLMODE desired for the call. If the option is not used, Load works in the default mode ADABAS.

xload -S ADABAS

Specifying a Command With a Call

In Load, command files can be started interactively or in batch mode. Calls are for the

1. interactive mode:

   xload -u parker,secret -d testdb -r filename

   Load executes the statements of the command file and then displays the input screen. If -P (PROMPT) was not specified, Load executes the indicated command file in NOPROMPT mode.

2. batch mode:

   xload -u parker,secret -d testdb -b filename

   In this case, Load suppresses any screen interaction and terminates after execution.

To execute the process in background, specify the corresponding shell command (&):

xload -b filename &

3. execution with parameter transfer

   xload -r filename 21.00 Mayr

   xload -b filename 21.00 Mayr

   In this example, the values "21.00" and "Mayr" are assigned to the formal parameters of the command file "filename". The blank has the effect of a separator between two parameters.

   All of these call formats can also be used from a shell script.

Load Protocol File

The protocol file written by Load is a normal file named "load.prot" stored in the directory from which Load was called. Name and path of the protocol file can be specified using the SET command. If the protocol file is to be output to the printer specified using the Set parameters, "PRINTER" (note the uppercases!) must be specified as filename.

Load Return Codes

When an error occurs, Load returns one of the following codes to the calling environment:

```
1: -8888 SERVERDB NOT ACCESSIBLE


2: -8000 SERVERDB MUST BE RESTARTED


3: -1021 TOO MANY USERS CONNECTED


4: -4008 UNKNOWN USER NAME/PASSWORD COMBINATION


5: Invalid call option. (The specified command is
                         not available to this tool.)


6: The protocol file cannot be created.


7: SQL error


8: Load error
```

```
 9: Rows rejected by DATALOAD or DATAUPDATE


10: File error in a statement
```

Remarks:

For LOAD BATCH, the return codes 1 to 6 implicitly mean that the job was not started. The return codes 7 to 10 are default return codes. LOAD terminates with one of these codes when the LOAD command file does not contain a statement to set a special return code. These values should be avoided when a return code is set by using the STOP or RETURNCODE statement.

End User Tool Query

As mentioned at the beginning there are two variants of Query available. The selection is done by the call command.

adquery

calls a GUI interface based on Tcl/TK. This does not yet support the full functionality of Query (compare the manuals "Query" and "GUI Query").

xquery

displays a character-oriented interface for an alphanumeric terminal.

This chapter covers the following topics:

- Calling Query
- Query Return Codes

Calling Query

Format:

Call:	Call:
xquery [<connect spec>]	adquery [<connect spec>]
[<commandfile spec>]	
| xquery [<connect spec>]	
[<QUERY command spec>]	
| xquery [<connect spec>]	
[<QUERY LIST option>]	
| xquery -V	| adquery -V
| xquery -h	| adquery -h
Call options:	Call options:
connect spec> ::=	<connect spec> ::=
[-U <user option>]	[-U <user option>]
[-u <user id> [,<password>]]	[-u <user id>[,<password>]]
[-d <serverdb>]	[-d <serverdb>]
[-n <servernode]	[-n <servernode]
[-I <isolation level>]	[-I <isolation level>]

[-t <session_timeout>] [-t <session_timeout>]

[-s]

[-S ADABAS] [-S ADABAS]

<commandfile spec> ::=

-r <filename>

[<parameter list>]

| -b <filename>

[<parameter list>]

<QUERY command spec> ::=

-R <stored command>

[<parameter list>]

| -B <stored command>

[<parameter list>]

| -e <object_name>,

<filename> [-A]

| -i <filename>

<QUERY LIST option> ::=

-L

Parameters: Parameters:

<user option> <user option>

::= <userkey> | prompt ::= <userkey> | prompt

<parameter list>

::= <parameter> <blank>

[<parameter list>]

<object_name>

::= <search name>*

| <stored command name>

Calling Query (general format)

adquery xquery

The options -u, -U, -d, and -n required for the connect are described in Section "Connect", the options -V and -h in Section "Adabas Tools: General Properties".

After the connect, the tool is in input mode where SQL statements can be entered.

Specifying an ISOLATION LEVEL

The ISOLATION LEVEL determines the read and write locks QUERY must use in certain situations. If no specification is made, ISOLATION LEVEL 0 is assumed. A description of the possible values and their meanings is contained in the "Reference" manual.

Specifying a TIMEOUT Value

The SESSION TIMEOUT value determines the time interval in seconds at the end of which the session will be terminated if it was not active. The database administrator can determine this value for the whole database using CONTROL or for a single user on his creation (default: 300 seconds). The option -t allows the user to specify a smaller value in seconds for the current session. A value larger than predefined produces an error message.

adquery -t 90	xquery -t 90

The database session started with this call is terminated after 90 seconds of inactivity

Calling Query in SELECT Mode

xquery -s

In SELECT mode, only read accesses to database objects can be performed. This mode is valid during the whole Query session.

Specifying an SQLMODE

The option -S can be used to specify the SQLMODE desired for the call. If the option is not used, Query works in the default mode ADABAS.

adquery -S ADABAS	xquery -S ADABAS

Specifying a Command File With a Call

In Query, command files can be started interactively or in batch mode. Calls are for the

1. interactive mode:

   xquery -u parker,secret -d testdb -r filename

   Query executes the statements of the command file and then displays the input screen.

2. batch mode:

   xquery -u parker,secret -d testdb -b filename

In this case, Query suppresses any screen interaction and terminates after execution.

To execute the process in background, specify the corresponding shell command (&):

adquery &

xquery -b filename &

The contents of the specified file are copied into the edit area and executed. The command file must therefore contain a sequence of SQL and report statements separated by comment lines. The command file must not exceed 12 KB (see the "Query" manual).

3. execution with parameter transfer

   xquery -r filename 21.00 Mayr

   xquery -b filename 21.00 Mayr

   In this example, the values "21.00" and "Mayr" are assigned to the formal parameters of the command file "filename". The blank has the effect of a separator between two parameters.

All of these call formats can also be used from a shell script.

Specifying a Command With a Call

In Query, stored commands can be started interactively or in batch mode. Calls are for the

1. interactive mode:

   xquery -u parker,secret -d testdb -R HOTEL

   Query executes the specified command and then displays the input screen.

2. batch mode:

   xquery -u parker,secret -d testdb -B HOTEL

   In this case, Query suppresses any screen interaction and terminates after execution.

   To execute the process in background, specify the corresponding shell command (&):

   xquery -B HOTEL &

3. execution with parameter transfer

   xquery -R command1 21.00 Mayr

   xquery -B command1 21.00 Mayr

   In this example, the values "21.00" and "Mayr" are assigned to the formal parameters of the stored command "COMMAND1". The blank has the effect of a separator between two parameters.

All of these call formats can also be used from a shell script.

Exporting or Importing Commands in Batch Mode

xquery -e HOTEL,hot.cmd

xquery -e HOT*,hot.cmd

xquery -i hot.cmd

In the first example, Query exports the stored command "HOTEL" into the Unix file "hot.cmd". In the second example, Query exports all stored commands whose name begin with "HOT". In the third example, Query imports all stored commands recorded in the Unix file "hot.cmd". In all cases, no screen interaction takes place. Query terminates after execution.

Specifying "-A" (APPEND) ensures that text is added at the end of an already existing file rather than overwriting the file.

To execute the process in background, specify the corresponding shell command (&):

xquery -i hot.cmd &

All of these call formats can also be used from a shell script.

Calling Query With the Query Command "LIST"

xquery -L

Connecting is done in a similar way to that described for the general Query call. The user does not access the input mode but the menu of the stored Query commands. The user can then execute any displayed command but cannot create new commands.

Query Return Codes

When an error occurs, Query returns one of the following codes to the calling environment:

```
1:  -8888 SERVERDB NOT ACCESSIBLE


2:  -8000 SERVERDB MUST BE RESTARTED


3:  -1021 TOO MANY USERS CONNECTED


4:  -4008 UNKNOWN USER NAME/PASSWORD COMBINATION


5: -13503 NAME OF STORED COMMAND MISSING


6: -13506 STORED COMMAND NOT FOUND
```

```
 7: -13508 PARAMETER LIST TOO SHORT OR MISSING

 8: SQL error

 9: -13523 INVALID REPORT COMMAND

10: Other errors
```

Adabastclsh and Adabaswish

These tools that are based on Tcl provide Adabas D with a command line oriented SQL interface. The calls

adabastclsh

adabaswish

start a fully functional Tcl shell. A prompt appears at which all valid Tcl and SQL statements can be entered. Adabastclsh outputs the data via the shell’s standard out, whereas Adabaswish starts a special TK-oriented window for the output.

When called with

adabastclsh <filename>

adabaswish <filename>

the tools work in batch mode. The specified file contains Tcl statements or "Adabas Tcl statements". More details about the syntax of these statements are included in the description of Adabastclsh and Adabaswish in the "User Manual Internet".

Programming Tool SQL-PL

The call of SQL-PL depends on whether the user intends

- to generate a new program or to modify an existing program via the SQL-PL workbench.
- to call an existing program via the SQL-PL interpreter.
- to make an existing program available to the end user or to install a program, both by means of the SQL-PL interpreter for the installation of applications.

This chapter covers the following topics:

- Call of the SQL-PL Workbench
- Call of the SQL-PL Interpreter
- Call of the SQL-PL Interpreter for Applications Installation
- Integration into the System Environment
- SQL-PL Return Codes

Call of the SQL-PL Workbench

The SQL-PL workbench is used to create, test, administer, and execute SQL-PL programs.

Call Syntax of the SQL-PL Workbench:

```
Call:


  xpl [<connect spec>] [<SQL-PL command spec>]


| xpl -V
| xpl -h



Call options:


  <connect spec>        ::=     [-U <user option> ]
                      [-u <user id>[,<password>]]
                      [-d <serverdb>] [-n <servernode]
```

```
                    [-t <session timeout>]
                    [-I <isolation level>]

  <SQL-PL command spec> ::=
       -R <SQL-PL object> [<parameter list>]
     | -B <SQL-PL object> [<parameter list>]
     | -e <object_name>,<filename> [ -A ]
     | -i <filename>



Parameters:

  <user option>     ::= <userkey> | prompt

  <isolation level> ::= 0 | 1 | 2 | 3 | 4

  <SQL-PL object>   ::= <owner>.<name1>.<name2>
                      | <name1>.<name2>
                      | <name1>

  <object_name>     ::= <name1>

  <parameter list>  ::= <parameter> blank [<parameter list>]
```

Call of the SQL-PL Workbench (general format)

xpl

After the connect the user is either in the application menu or, if he does not have any SQL-PL modules and no call privileges for the programs of other users, in the editor.

Specifying a TIMEOUT Value

In addition to the user specifications USERID, PASSWORD, SERVERDB and SERVERNODE which can be made for the call of an Adabas tool via call options (see Section 2), a SESSION TIMEOUT value specified. In this way the user can reduce the time interval at the end of which the session will be terminated if it was not active for a certain period of time.

xpl -t 90

The database session started via this call iserminated after 90 seconds of inactivity

Specifying the ISOLATION LEVEL

If the user wants to work in a particular ISOLATION LEVEL, then he can specify it with the SQL-PL call. Possible are the specifications 0, 1, 2, 3, or 4 (see the "SQL-PL" manual). Example:

xpl -d testdb -I 2

Specifying a Command With a Call

When calling SQL-PL there is the possibility of starting programs either interactively or in batch mode as well as of exporting or importing programs in batch mode.

The program has to contain a module named ’start’ in order that a program is capable of being started only by specifying its name. If a module with such a name is not available, the name of the program to be called as the first module has to be specified in addition to the program name.

Note: SQL-PL does not distinguish between program names in upper or lower case characters.

1. Call of a User’s own SQL-PL Program

   - in interactive mode:

     xpl -R HBL

     xpl -R HBL.start

     xpl -u parker,secret -d testdb -R HBL

     SQL-PL executes the program HBL and then terminates the database session.

   - in batch mode:

     xpl -B turnover

     xpl -B turnover.start

     xpl -u parker,secret -d testdb -B list.print

     in this case the corresponding program is executed without any screen interaction.

     Note:
     Programs can only be called in batch mode, when they are suited for this mode. Programs where input and output is made via the screen while being executed cannot be called in batch mode. If the attempt is made to perform such a program in batch mode, a corresponding error message is output.

   - in batch mode as background process:

If the user wants the SQL-PL program to be executed as background process, he has to specify the corresponding shell command (&):

- xpl -B turnover &
- xpl -B list.print &

2. Call of Other Users' SQL-PL Programs

   If a program is to be started which belongs to the library of another user <owner> but for which the current user has got the call privilege, then the owner name and the start module (even if the module is named 'start') have to be specified in addition to the program name. Examples:

   xpl -u parker,secret -R george.HBL.start or

   xpl -u parker,secret -B george.TEST.start

3. Executing a Program Passing Parameters

   xpl -R prog1.xa 21.00 Mayr

   In this example the values '21.00' and 'Mayr' are assigned to the formal parameters of the module 'xa' of the program 'prog1'.

   The blank has the effect of a separator between two parameters.

   If character strings containing blanks or quotes are to be passed as parameters, they have to be enclosed in quotes (single or double). These on their part have to be protected against the Unix shell interpreter by a '\' (backslash).

   Examples:

   xpl -R georg.proj1.start \"Say Hallo\"

   In this example a parameter with the value 'Say Hallo' is passed.

   xpl -R georg.proj1.start \"Say \'Hallo\'\"\'\'\"what\'s the matter\"

   Here three parameters are passed to the program:

   1. Say 'Hallo'
   2. ' ', which is interpreted as NULL
   3. what's the matter

4. Export/Import of Programs

   xpl -e HBL hbl.appl

   The program HBL is written to an operating system file named 'hbl.appl'.

xpl -i hbl.appl

The program is transferred from the operating system file ’hbl.appl’ to the user’s own SQL-PL program library.

All these call formats can also be used from a shell script which can be called by its name.

Call of the SQL-PL Interpreter

The SQL-PL interpreter is intended especially for end users who only execute programs of other users and do not have a program library of their own. The SQL-PL interpreter can be called, like the SQL-PL workbench, call options (see "Call of the SQL-PL Workbench").

The SQL-PL interpreter is called in the following way:

xplrun

The execution of a program <prog name> of the user’s own library can be started from the shell or a shell script with the call

xplrun -R <prog name>.<module name>

If the program has a module named ’start’, then the call

xplrun -R <prog name>

is sufficient.

If a program is to be started which belongs to the library of another user <owner> and for which the current user has got the call privilege, then the call always runs as follows (even if the module is named ’start’):

xplrun -R <author>.<prog name>.<module name>

If all connect parameters are known to Adabas, the program is immediately executed, otherwise a connect screen displayed into which the connect parameters unknown to Adabas have to be entered first.

Note: The SWITCH construct of SQL-PL allows an appropriate program menu to be defined for every end user who can do then with a single XPLRUN call.

Call of the SQL-PL Interpreter for Applications Installation

In addition to the possibility of executing SQL-PL programs the SQL-PL interpreter for the installation of applications provides administrative functions which are necessary to make an application accessible to a particular user. For these tasks a workbench is available which, except for the functions for constructing and testing a program, provides the user with the full workbench functionality, in particular with:

- all show commands
- all authorizing commands (PRIVILEGES, EXPORT, IMPORT, GRANT, REVOKE)

- the installation command IMPORT <filename>
- the SET command
- VERSION (for the display of the workbench version)
- HELP

The SQL-PL interpreter for the installation of applications is called in the following way:

xtplrun

XTPLRUN can be called like XPL and XPLRUN with call options (see "Call of the SQL-PL Workbench" and "Call of the SQL-PL Interpreter").

Integration into the System Environment

The workbench functions IMPORT/EXPORT do not only build bridges between the SQL-PL libraries but also between the SQL-PL library and the Unix file system. The command

==> export customer customer.appl

writes all modules of the program ’customer’ one after the other to the Unix file ’customer.appl’. The individual modules are separated from each other by a line which only contains the keyword ENDMODULE.

On the other hand a Unix file having the same structure as a file generated with EXPORT can be read into the workbench by means of the command

==> import customer.appl

Thereby each single module is implicitly checked by means of STORE and already existing modules with the same name are overwritten. This feature enables the user in particular

- to generate and maintain complete programs in Unix files by means of his accustomed system editor - the vi for example.
- to temporarily save programs in private Unix files.
- to apply the mechanisms which are used to administer and file program versions (Cobol, Pascal) to the SQL-PL programs as well.

SQL-PL Return Codes

When an error occurs, the following codes are returned to the calling environment:

```
 1: -8888 SERVERDB NOT ACCESSIBLE


 2: -8000 SERVERDB MUST BE RESTARTED
```

```
 3: -1021 TOO MANY USERS CONNECTED


 4: -4008 UNKNOWN USER NAME/PASSWORD COMBINATION


 5: Invalid call option
    The specified command is not available to this component.


 6: The command is not available for standard users.
    Standard users are only allowed to execute existing SQL-PL
    programs of other users. They are not able to create or
    import any programs.


 7: Workbench command resulted in an error.
    The specified command could not be terminated successfully
    (e.g. translation error, file could not be opened).


 8: SQL-PL program terminated with runtime error.
```

When setting return codes by means of the STOP statement, the value specified here should be avoided, if possible.

C / C++ Precompiler

This chapter covers the following topics:

- C/C++ Precompiler Calls and Options
- Compiling the Precompiled C/C++ Program
- Linking the Compiled C/C++ Program
- Executing the Linked C/C++ Program
- C/C++ Precompiler Runtime Options
- C/C++ Precompiler Input/Output Files
- Operating System Commands
- C/C++ Precompiler Include Files

C/C++ Precompiler Calls and Options

cpc <precompiler options> <fn> <compiler options>

<fn> ::= file name

The name of the source code file must be <fn>.cpc.

C/C++ precompiler options:

```
ansi c          ::=    -E cansi
c++                    ::=     -E cplus
cachelimit          ::=        -y <cache limit>
check nocheck          ::=     -H nocheck      (Default: -H check)
check syntax           ::=     -H syntax
comment                ::=     -o
compatible          ::=        -C
datetime europe        ::=     -D eur
datetime iso      ::=  -D iso  (Default: -D internal)
datetime jis           ::=     -D jis
datetime usa           ::=     -D usa
extern          ::=    -e
help              ::=  -h
```

```
isolation level        ::=     -I <isolation level>    (Default: -I 10)

list              ::=  -l

margins         ::=    -m <lmar,rmar>  (Default: -m 1,132)

nowarn          ::=    -w

precom                 ::=     -c

profile                ::=     -R

program                ::=     -P <progname>   (Default: -P  <filename>)

serverdb              ::=      -d <serverdb>

servernode             ::=     -n <servernode>

silent          ::=    -s

sqlmode adabas  ::=    -S adabas       (Default: -S adabas)

timeout                ::=     -t <timeout>

trace file             ::=     -F <tracefn>

trace long             ::=     -X

trace short            ::=     -T

user              ::=  -u <usern>,<passw>

userkey                ::=     -U <userkey>

version                ::=     -V
```

For an explanation of the different precompiler options, see the

"C/C++ Precompiler" manual.

Compiler option: see the compiler manual

Set is: -c

Sample call: cpc -u DBUSER,DBPWRD test

Additional connect data is fetched for the corresponding session from the connect command specified in the program and/or from the ADUSER file. If no ADUSER file is available, all connect data must be specified using the precompiler options. These options are only valid for session 1.

Compiling the Precompiled C/C++ Program

cl compiler options <fn>.c

A source file saved after its precompilation can be compiled in the usual way with cc. All compiler options are allowed. -c -I$DBROOT/incl is the default option for the cc call implicitly made by cpc.

Supporting lint

cpclint [-S <sqlmode>] [lint options] <fn>.c ... [<llib>.ln ...]

The shell script cpclint supports the checking of precompiled "*.c" files by lint. The specified <sqlmode> must be identical to the -S option of the precompiler run. The default is Adabas. All lint options can be specified. The include directory $DBROOT/incl and the lint libraries are set implicitly. The libraries are $DBROOT/lib/llib-lpcr.ln for -S adabas, $DBROOT/lib/llib-lora.ln for -S oracle and $DBROOT/lib/llib-ldb2.ln for -S db2 or -S ansi. Own lint libraries can be specified as "*.ln" files.

Linking the Compiled C/C++ Program

An Adabas application program is linked with the shell script cpclnk. The library files needed are stored in the $DBROOT/lib directory. Their names are output when calling cpclnk.

cpclnk <fn> <fn1> ...

The file name of the main program must be specified as first parameter. Then "<fn>.o" or "<fn>.c" is expected as the input file, and the executable file "<fn>" is created as the output of the linkage editor. All the other file parameters can be object files "*.o", source files "*.c" or libraries "*.a" specified in any order.

Example: cpclnk fn fn1 fn2 fn3

fn.c, fn1.o, fn2.a, fn3.o are available.

The executable program receives the name fn.

Executing the Linked C/C++ Program

Options are passed to the program in the shell variable SQLOPT. If the option -k is set in the current shell (e.g., using set -k), SQLOPT can be transferred as keyword parameter:

Example:

<fn> SQLOPT="-X -d MyDatabase"

or

SQLOPT="-X -d MyDatabase" <fn>

Enter this command to execute the linked program.

C/C++ Precompiler Runtime Options

```
cachelimit            ::= -y <cache limit>

isolation level       ::= -I <isolation level>
```

```
mfetch                ::= -B <number>

no select direct fast ::= -f

profile               ::= -R

serverdb              ::= -d <serverdb>

servernode            ::= -n <servernode>

timeout               ::= -t <timeout>

trace alt             ::= -Y <statement count>

trace file            ::= -F <tracefn>

trace long            ::= -X

trace no date/time    ::= -N

trace short           ::= -T

trace time            ::= -L <seconds>

user                  ::= -u <usern>,<passw>

userkey               ::= -U <userkey>
```

For an explanation of the different precompiler options, see the

"C/C++ Precompiler" manual.

C/C++ Precompiler Input/Output Files

<fn>.pcl:	Precompiler source and error listing.
sqlerror.pcl:	Adabas error file. This file is output when errors occur before the file "<fn>.pcl" has been opened.
<fn>.o:	Object module. Linked to an executable module with other object modules and the runtime system.
<fn>.lst:	Compiler source and error listing.
<fn>.pct:	Trace file. It contains the performed SQL statements.
<fn>.c:	The precompiled application program.
<fn>.w1:	Precompiler work file.
<fn>.w2:	Precompiler work file.
<fn>.w3:	Precompiler work file.

Operating System Commands

Unix shell commands and executable programs can be called from source code using the "exec command ..." (see Section "Calling Operating System Commands").

Examples:

<command> := ’ ls -l ’	displays the current directory
<command> := ’ lp out ’	prints the file "out"
<command> := ’ pgm1 > pgm1.lis’	starts the program "pgm1" writing the results to the file "pgm1.lis".

C/C++ Precompiler Include Files

The precompiler generates the preprocessor directive #include $DBROOT/incl/cpc.h. This file contains all declarations required for the translation of a C/C++ program.

Cobol Precompiler

This chapter covers the following topics:

- Special Features
- Cobol Precompiler Calls and Options
- Compiling the Precompiled Cobol Program
- Linking the Compiled Cobol Program
- Executing the Linked Cobol Program
- Cobol Precompiler Runtime Options
- Cobol Precompiler Input/Output Files
- Operating System Commands
- Cobol Precompiler Include Files
- The Cobol Precompiler for ACU Cobol

Special Features

The Microfocus Cobol compiler is used as the default compiler. It is called with "/usr/bin/cob".

If the Microfocus Cobol compiler is to be called from another directory or if a Cobol compiler of another manufacturer is used, the call must be entered in the shell variable SQLCOBCOM. This call overrides the default call.

Example: SQLCOBCOM="$COBDIR/bin/cob"

Subroutine for Entering a Variable’s Address

For the usage of the DESCRIBE statement, a variable’s address must be entered into the SQLDA. For this purpose, the subroutine "sqbaddr" is distributed together with the precompiler runtime system.

Cobol Precompiler Calls and Options

cobpc <precom options> <fn> <compiler options>

<fn> ::= filename

The complete name of the precompiler input file (source) must be <fn>.cobpc.

Cobol pecompiler options:

```
CACHELIMIT           ::=       -Y <cache limit>

CHECK NOCHECK          ::=     -H nocheck      (Default: -H check)

CHECK SYNTAX       ::= -H syntax

COBOL-DIALECT          ::=     -E ANSI85       (Default: -E Micro Focus)

COMMENT                 ::=    -o

COMPATIBLE           ::=       -C

DATE-TIME EUROPE       ::=     -D eur

DATE-TIME ISO     ::=  -D iso  (Default: -D internal)

DATE-TIME JIS     ::=  -D jis

DATE-TIME USA     ::=  -D usa

DECPOINT COMMA         ::=     -p      (Default:  POINT)

EXTERN           ::=   -e

HELP               ::= -h

ISOLATION LEVEL         ::=    -I <isolation level>    (Default: -I 10)

LIST               ::= -l

MARGINS                 ::=    -m <mbegin>,<mend>      (Default: -m 1,72)

NOWARN           ::=   -w

PRECOM                 ::=     -c

PROFILE                 ::=    -R

PROGRAM                ::=     -P <progname>   (Default: -P  <filename>)

QUOTE  DOUBLE     ::=  -q      (Default: SINGLE)

SERVERDB               ::=     -d <serverdb>

SERVERNODE             ::=     -n <servernode>

SILENT           ::=   -s

SQLMODE ADABAS   ::=   -S adabas       (Default: -S adabas)

TIMEOUT          ::=   -t <timeout>

TRACE FILE             ::=     -F <tracefn>

TRACE LONG       ::=   -X

TRACE SHORT            ::=     -T

USER               ::= -u <usern>,<passw>
```

```
USERKEY                 ::=    -U <userkey>

VERSION                ::=     -V
```

For an explanation of the different precompiler options, see the

"Cobol Precompiler" manual.

The option "-i" has the effect that the lines in the precompiler listing are correctly counted with relation to the source file. This option must be set when the program was translated with XMS before it was precompiled with the Adabas precompiler.

Compiler options: see the compiler manual

Sample call: cobpc -u DBUSER,DBPWRD test -o sqldbtest

Additional connect data is fetched for the corresponding session from the connect command specified in the program and/or from the ADUSER file. If no ADUSER file is available, all connect data must be specified using the precompiler options. These options are only valid for session 1.

Compiling the Precompiled Cobol Program

cobolpc <compiler options> <fn> <external objects>

Options: see the compiler manual

Specifying the option -c creates an output file <fn>.cob. This is the input for the Cobol compiler. It can only be compiled using the call "cobolpc".

Linking the Compiled Cobol Program

An Adabas application program is linked with the shell script cobpclnk. The library files needed are stored in the $DBROOT/lib directory. Their names are output when calling cobpclnk.

cobpclnk <main> <external objects or libs>

The file name of the main program <main> must be specified as the first parameter. The executable program takes the name of the file (without a suffix). The other file parameters are also listed without a suffix and must be object modules "*.o" or libraries "*.a" in any sequence.

Example: cobpclnk test fn1 fn2

There are test.o, fn1.a, fn2.o.

The executable program receives the name test.

Linking a Cobol Runtime Module with Adabas

cobpcrts <rts name> <external objects or libs>

This shell script can be used to generate a private Cobol runtime module which executes Adabas applications.

Call:

<rts name> <fn>

where <fn> must be ".int" or ".gnt".

Executing the Linked Cobol Program

Options are passed to the program in the shell variable SQLOPT. If the option -k is set in the current shell (e.g., using set -k), SQLOPT can be transferred as a keyword parameter.

Example:

<fn> SQLOPT="-X -d MyDatabase"

or

SQLOPT="-X -d MyDatabase" <fn>

Enter the filename to execute the linked program.

Cobol Precompiler Runtime Options

```
CACHELIMIT     ::=     -Y <cache limit>

ISOLATION LEVEL        ::=     -I <isolation level>

MFETCH ::=     -B <number>

NO SELECT DIRECT FAST  ::=     -f

PROFILE        ::=     -R

SERVERDB       ::=     -d <serverdb>

SERVERNODE     ::=     -n <servernode>

TIMEOUT        ::=     -t <timeout>

TRACE ALT      ::=     -Y <statement count>

TRACE FILE     ::=     -F <tracefn>

TRACE LONG     ::=     -X

TRACE NO DATE/TIME     ::=     -N

TRACE SHORT    ::=     -T

TRACE TIME     ::=     -L <seconds>

USER   ::=     -u <usern>,<passw>

USERKEY        ::=     -U <userkey>
```

For an explanation of the different precompiler options, see the

"Cobol Precompiler" manual.

Cobol Precompiler Input/Output Files

<fn>.pcl:	Precompiler source and error listing.
sqlerror.pcl:	Adabas error file. This file output when errors occur before the file "<fn>.pcl" has been opened.
<fn>.o:	Object module. Linked to an executable module with other object modules and the runtime system.
<fn>.lst:	Compiler source and error listing.
<fn>.pct:	Trace file. It contains the performed SQL statements.
<fn>.cob:	The precompiled application program.
<fn>.pass1:	Precompiler work file.
<fn>.w1:	Precompiler work file.
<fn>.w2:	Precompiler work file.
<fn>.w3:	Precompiler work file.
<fn>.lp1:	Precompiler work file.
<fn>.ln1:	Precompiler work file.

Operating System Commands

Unix shell commands and executable programs can be called from source code using the "exec command ..." (see Section "Calling Operating System Commands").

Examples:

<command> := ’ ls -l ’	displays the current directory
<command> := ’ lp out ’	prints the file "out"
<command> := ’ pgm1 > pgm1.lis’	starts the program "pgm1" writing the results to the file "pgm1.lis".

Cobol Precompiler Include Files

The include files are stored in the file directory $DBROOT/incl. Their names begin with "csql", e.g., "csqlca.i". These files must not be modified because they are required for an application to run. The following include files may also be used as copy elements by the user:

CSQLDA

contains the elements of the SQLDA with 300 SQLVAR entries and can, for example, be used in a declaration in the following way:

```
01  SQLDA1.

    COPY "CSQLDA".
```

CSQL1DA

contains the main structure of the SQLDA (without SQLVAR entries)

CSQL2DA

contains the SQLVAR elements. The include files can be used in the following way, for example:

```
01  SQLDA2.

    COPY "CSQL1DA".

    02 SQLVAR OCCURS 10 TIMES.

              COPY "CSQL2DA".
```

Thus the number of SQLVAR entries is defined within the program.

The following include files are used for Oracle compatibility. They contain the Oracle descriptors in the variant required by Adabas.

CSQLOR1

contains SET statements for assigning the addresses required within the bind descriptor (BNDDSC). This include file must be inserted into the Procedure Division before the first SQL statement.

CSQLOR2

contains the declaration of the bind descriptor (BNDDSC) for up to 300 entries.

CSQLOR3

contains the declaration of the select descriptor (SELDSC) for up to 300 entries.

CSQLOR4

contains the SET statements for assigning the addresses required within the select descriptor (SELDSC). This include file must be inserted into the Procedure Division before the first SQL statement.

CSQLADR

contains a Cobol subroutine for determining a variable’s address. This subroutine must be called in the following way:

```
CALL "SQLADR" USING <variable name> <pointer variable>
```

The Cobol Precompiler for ACU Cobol

acupc <precom options> <fn> <compiler options>

<fn> ::= filename

The complete name of the precompiler input file (source) must be <fn>.cobpc.

As the ACU Cobol compiler is implicitly executed, the programmer must ensure that the copy elements can be found. For this purpose, either the shell variable

COPYPATH=$DBROOT/incl

must be set or the following compiler option

-Sp $DBROOT/incl

be specified-

Sample call: acupc -H nocheck test -Sp $DBROOT/incl

Linking the ACU Cobol runtime module (runcbl)

In $DBROOT/incl there is a "sub85.c" as a pattern. The original "sub85.c" must be changed using the files "sub85def.h" and "sub85fun.h". Moreover, there is a "makefile" available as a pattern that can be used to change the original makefile.

Options are passed to the program in the shell variable SQLOPT .

Example:

SQLOPT="-X -d MyDatabase" runcbl <fn>.out

Call Interface (ODBC)

The Adabas ODBC driver enables access to the Relational Database Management System Adabas D on a server.

Under Unix, the ODBC driver is realized as static library that is linked to the application program. The functions to be used are described in the "User Manual ODBC".

This chapter covers the following topics:

- Translating an ODBC Application
- Linking an ODBC Application
- Executing an ODBC Application and Runtime Options
- Files

Translating an ODBC Application

The include path must be completed. The required include files are stored in the $DBROOT/incl directory. An example makefile containing all specifications including the linker call odbclnk (that is described in Section "Linking an ODBC Application") is stored in the $DBROOT/demo/eng/ODBC directory.

Linking an ODBC Application

To link an ODBC application program for Adabas, the shell script odbclnk is used. The library files needed for ODBC and the Adabas runtime environment are stored in the $DBROOT/lib directory. Their names are output when calling odbclnk.

odbclnk <options> <main> <external objects or libs>

The file name of the main program <main> must be specified as the first parameter after the linker options. The executable program receives the name of the file (without suffix). The other file parameters are also noted without suffix and must be object modules "*.o" or libraries "*.a" in any sequence.

odbclnk [-b] <option> <main> <external objects or libs>

The option [–b] allows dynamic library files ".so" to be linked to the main program.

These files are also stored in the $DBROOT/lib directory. The dynamic library files are load runtime of the main program and must be available in the library directory.

Example: odbclnk test fn1 fn2

There are test.o, fn1.a, fn2.o .

The executable program receives the name test.

Executing an ODBC Application and Runtime Options

Runtime options for an ODBC application are not passed using the environment variable SQLOPT (except the trace options, see below in this section). They are contained in the /usr/spool/sql/config/ODBC.ini file. For a description of all runtime options and the format of the ODBC.ini file see the "User Manual ODBC".

User specifications that are provided in an ADUSER file cannot be used in an ODBC application. The environment variables SERVERDB and LOCALE are not evaluated either. To be able to execute an ODBC application, it must contain the user specifications required for the opening of a database session in a corresponding ODBC function (see the "User Manual ODBC", Section "Supported Functions"). These specifications, however, can be overridden by entries provided in the ODBC.ini file.

The following options can be used to control the trace output of an ODBC application:

trace alt	::=	-Y <statement count>
trace file	::=	-F <tracefn>
trace long	::=	-X
trace no date/time	::=	-N
trace short	::=	-T
trace time	::=	-L <seconds>

These options are passed in the environment variable SQLOPT .

Example:

The command

SQLOPT="-X -F mytrace" <program name>

calls the program, and when executing it, writes a long SQL trace into the file "mytrace".

For the exact meaning of the trace options, see the "User Manual ODBC".

Files

When using the ODBC interface for Adabas under Unix, the following files are required:

In the $DBROOT/lib directory:

odbclib.a	ODBC driver library
libsql*.a	Adabas runtime environment library
odbclib.so	ODBC driver library (dynamic)
libsql*.so	Adabas runtime environment library (dynamic)

In the $DBROOT/incl directory:

sql.h

sqlext.h

WINDOWS.H

In addition, the $DBROOT/demo/eng/ODBC directory contains two example files for ODBC applications (sqlexamp.c and sqladhoc.c) and a makefile that can be used to translate applications.

Call Interface (JDBC)

System prerequisite for the usage of the JDBC interface is a JAVA interpreter. As JAVA is independent of a system there is no need to describe special Unix-specific properties. A more detailed description of the JDBC interface is contained in the "User Manual Internet".

A sample program showing the most important information on the structure of a JAVA database application is included in the directory DBROOT/demo/eng/JDBC.

Call Interface (OCI)

C applications can also access an Adabas database using the Oracle Call Interface (OCI). An existing application that uses the OCI Interface can work with Adabas without changes in the source files. Just a link to the Adabas OCI libraries is required. Differences between the OCI Interface and Oracle’s OCI which may force changes to the source files are described in Section "Special Remarks".

A description of the OCI entries can be found in the respective Oracle user manuals.

This chapter covers the following topics:

- Translating an OCI Application
- Linking an OCI Application
- Executing a Linked OCI Application
- Runtime Options
- The Trace File
- Profiling
- Special Remarks

Translating an OCI Application

The include path must be added. The OCI-specific include files are stored in the $DBROOT/incl directory.

Linking an OCI Application

The shell script "ocilnk" is used to link an OCI application for Adabas. The library files required for the OCI interface as well as for the Adabas runtime environment are stored in the $DBROOT/lib directory. Their names are output when calling "ocilnk".

ocilnk <options> <main> <external objects or libs>

Options: see the Linker manual

The filename of the main program <main> must be specified as first parameter after the linker options. The executable program receives the name of the file (without suffix). All the other file parameters are also noted without suffix and must be object modules "*.o" or libraries "*.a" in any sequence.

Example: ocilnk test fn1 fn2

There are test.o, fn1.a, fn2.o.

The executable program receives the name test.

Executing a Linked OCI Application

Options are passed to the program in the shell variable SQLOPT. If the option -k is set in the current shell (e.g., using set -k), SQLOPT can be transferred as keyword parameter:

Example:

<fn> SQLOPT="-X -d MyDatabase"

or:

SQLOPT="-X -d MyDatabase" <fn>

Entering this command executes the linked program.

Runtime Options

```
cachelimit                ::=  -Y < cache limit>

isolation level        ::=     -I <isolation level>

profile                   ::=  -R

serverdb                 ::=   -d <serverdb>

servernode             ::=     -n <servernode>

timeout                   ::=  -t <timeout>

trace alt          ::= -Y <statement count>

trace file             ::=     -F <tracefn>

trace long             ::=     -X

trace no date/time     ::=     -N

trace short            ::=     -T

trace time             ::=     -L <seconds>

user                 ::=       -u <usern>,<passw>

userkey                   ::=  -U <userkey>
```

For an explanation of the different precompiler options, see the

"C/C++ Precompiler" manual.

The Trace File

The trace file shows the executed OCI entries , including their actual parameters and information sent to or received from the interface to the Adabas kernel. The SQL statements are only recorded for parse requests to the kernel. The parse identification (parseid) can be used to find out which SQL statement is currently executed.

The information contained in the trace file depends on the trace option.

For a simple trace (TRACE SHORT), only the sequence of the executed OCI entries – including their actual parameters, the SQL statements sent to the database kernel and the return code of the OCI entries – are recorded.

Example:

```
_____________________________________________________________________

|                                                                   |
|=================================================================|
|=ORLON (00410668,004106a8,00400fdd,5,00400fe3,-1,0)                |
|SESSION  :  1                                                      |
|SQLMODE  :  ORACLE                                                 |
|SERVERDB :  ADB                                                    |
|SERVERNODE: adanode                                                |
|CONNECT "DEMO              " IDENTIFIED BY :A SQLMODE ORACLE       |
|=================================================================|
|=OOPEN (004107a8,00410668,00000000,-1,-1,00000000,-1)              |
|=================================================================|
|=OOPEN (004107e8,00410668,00000000,-1,-1,00000000,-1)              |
|=================================================================|
|=OCOF  (00410668)                                                  |
|=================================================================|
|=OPARSE(004107a8,00401072,-1,1,2)                                  |
|MDECLARE SQL_CUR00000000     CURSOR FOR SELECT NVL(MAX(empno),0)   |
|FROM emp                                                           |
|PARSE:   000014DBD00000013D012d00                                  |
|DESCRIBE                                                           |
|=================================================================|
```

```
|=ODEFIN(004107a8,1,7ffffb24,4,3,-1,                                |
|        00000000,00000000,-1,-1,00000000,00000000)                 |
|=================================================================== |
|=OEXFET(004107a8,1,0,0)                                            |
|EXECUTE: 000014DBD00000013D012d00                                   |
|RESULTTABLE: SQL_CUR00000000                                        |
|ROWCOUNT: 0                                                         |
|MFETCH SQL_CUR00000000    INTO :A                                   |
|PARSE:   000014DBD01000013D002b00                                   |
|EXECUTE: 000014DBD01000013D002b00                                   |
|ROWNO****   1                                                      |
|ROWCOUNT: 1                                                         |
|____________________________________________________________________|
```

If the detailed form of the trace file is specified (TRACE LONG), the input and output values of the SQL parameters involved and the execution time of the statements are included.

Example:

```
_____________________________________________________________________________

|                                                                           |
|=OOPEN (004107e8,00410668,00000000,-1,-1,00000000,-1)                      |
|=================================================================          |
|=OCOF  (00410668)                                                          |
|=================================================================          |
|=OPARSE(004107a8,00401072,-1,1,2)                                          |
|MDECLARE SQL_CUR00000000     CURSOR FOR SELECT NVL(MAX(empno),0) FROM emp  |
|PARSE:   000014DBD00000013D012d00                                          |
|START  :  DATE :  2002-06-14    TIME :  0012:26:59                         |
|END    :  DATE :  2002-06-14    TIME :  0012:26:59                         |
|DESCRIBE                                                                   |
|=================================================================          |
|=ODEFIN(004107a8,1,7ffffb24,4,3,-1,                                        |
|        00000000,00000000,-1,-1,00000000,00000000)                          |
|=================================================================          |
```

```
|=OEXFET(004107a8,1,0,0)                                                    |
|EXECUTE: 000014DBD00000013D012d00                                          |
|RESULTTABLE: SQL_CUR00000000                                               |
|ROWCOUNT: 0                                                                |
|START  :  DATE :  2002-06-14    TIME :  0012:26:59                         |
|END    :  DATE :  2002-06-14    TIME :  0012:26:59                         |
|MFETCH SQL_CUR00000000    INTO :A                                          |
|PARSE:   000014DBD01000013D002b00                                          |
|START  :  DATE :  2002-06-14    TIME :  0012:26:59                         |
|END    :  DATE :  2002-06-14    TIME :  0012:26:59                         |
|EXECUTE: 000014DBD01000013D002b00                                          |
|ARR-CNT**   1                                                              |
|ROWNO****   1                                                             |
|OUTPUT :    1: PARAMETER                      :  8294                      |
|ROWCOUNT: 1                                                                |
|START  :  DATE :  2002-06-14    TIME :  0012:26:59                         |
|END    :  DATE :  2002-06-14    TIME :  0012:26:59                         |
|___________________________________________________________________________|
```

The option TRACE ALT has the effect that the trace output is made alternately to two files. When doing so, as many executed OCI entries are recorded in each file as are specified in <statement count>. If there are more OCI entries to be executed than indicated by <statement count>, the files will be overwritten cyclically. The trace files are named "OCITRAC.pct" and "OCITRA2.pct". The long form of the trace is generated.

If the trace output should not be given the default name, the option TRACE FILE can be used to specify another name. If no other trace option was specified, the option TRACE SHORT is simultaneously enabled as a default. The filename must be standardized according to the operating system conventions. The name can also be specified as a character string constant (optional). The option overrides the option passed during the precompiler run. Only trace files with default trace filenames are not buffered on output.

The option TRACE NO DATE/TIME can be used to suppress the output of the date and time values for the start and end of the execution of an OCI entry made into the trace file.

With the option TRACE TIME, only those OCI entries are recorded whose execution time is greater than or equal to <seconds>. The long form of the trace is generated.

Profiling

When the option PROFILE is enabled during an application’s run, the Adabas kernel generates statistics on the processed OCI entries.

For every order, the date of runtime, number of calls and accumulated realtime is entered into the SYSPROFILE table of the local SYSDBA. The realtime consists of the time taken by the processing of a statement within an application program including all data conversions and time needed by the Adabas kernel. The time needed to enter this information into the SYSPROFILE table, however, is not included. The key of a row consists of the following specifications: user name, program name, module name, language of the application program, and line number of the statement within the application program related to the source and the internal parseid. With the enabled TRACE option, the time required for writing the trace to a file affects the profiling. Therefore, it is not convenient to activate the PROFILE and TRACE options at the same time.

The entries to the SYSPROFILE table are made within the transactions of the application program. Therefore, they are only stored in the table when the application program issues a COMMIT WORK.

When the option is enabled, old entries made for username, program name, and language are always deleted at the beginning of the program.

Special Remarks

The subroutine calls "sqlld2" and "oopt" have no effect.

"odessp" provides a description of the parameter specified Adabas Stored Procedure.

In certain situations, Adabas return codes can be passed to the application in addition to Oracle-compatible return codes.

The ROWID in the CURSOR DATA area is not set to a value.

Restrictions concerning Oracle SQL are described in the "Reference/Oracle" manual.

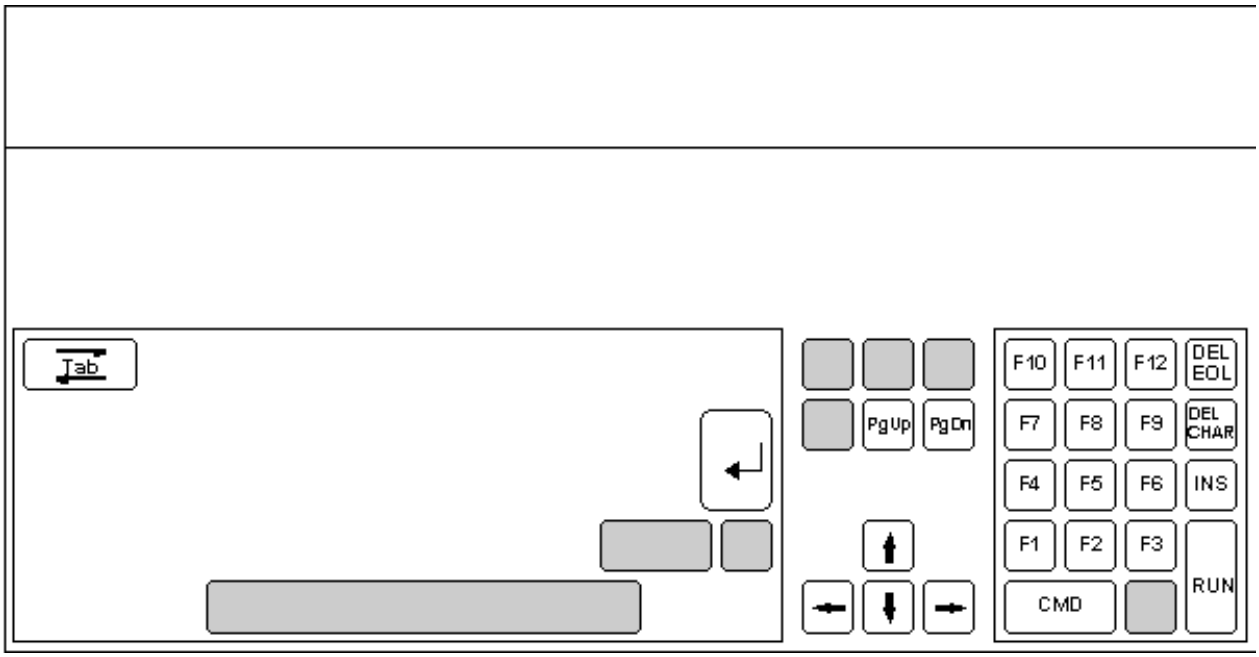
Call Interface (Perl)

In this way a Perl programmer is also provided with an Adabas interface. System prerequisite for its usage is a Perl interpreter. As Perl is independent of a system, there is no need to describe special Unix-specific properties.

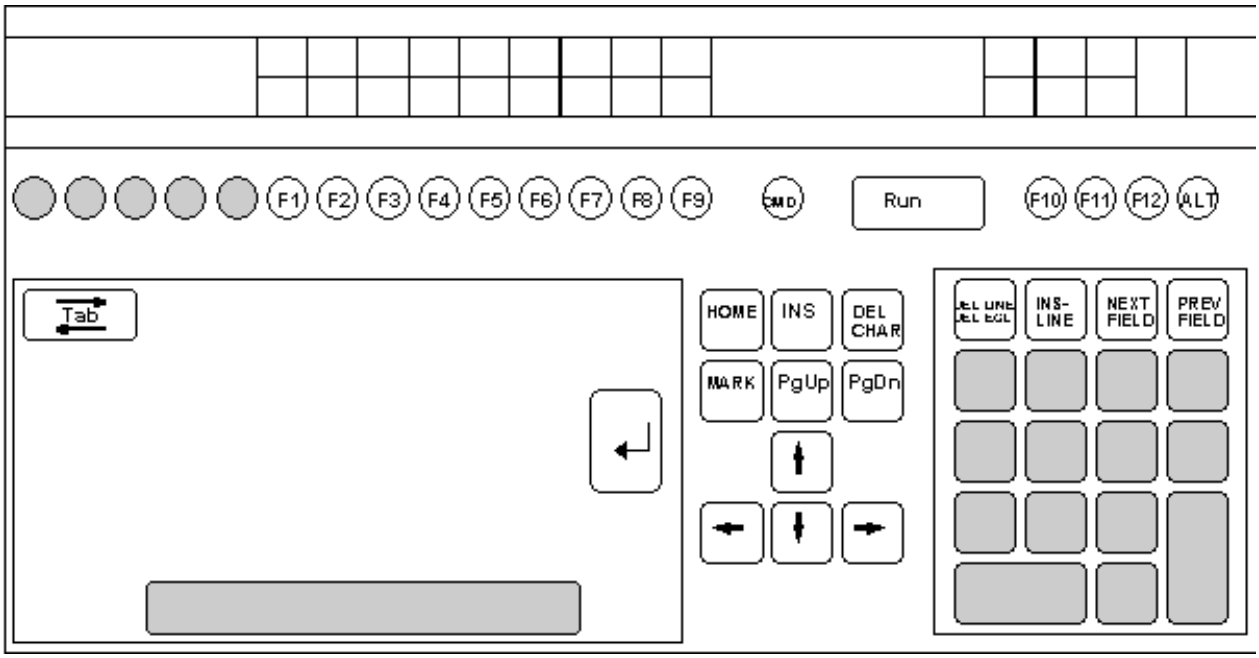
Adabas supports the Perl interface DBI which is independent of database systems as well as an interface which is particularly tailored for Adabas D. A more detailed description of these interfaces is contained in the "User Manual Internet".

Appendix - Keyboard Layouts

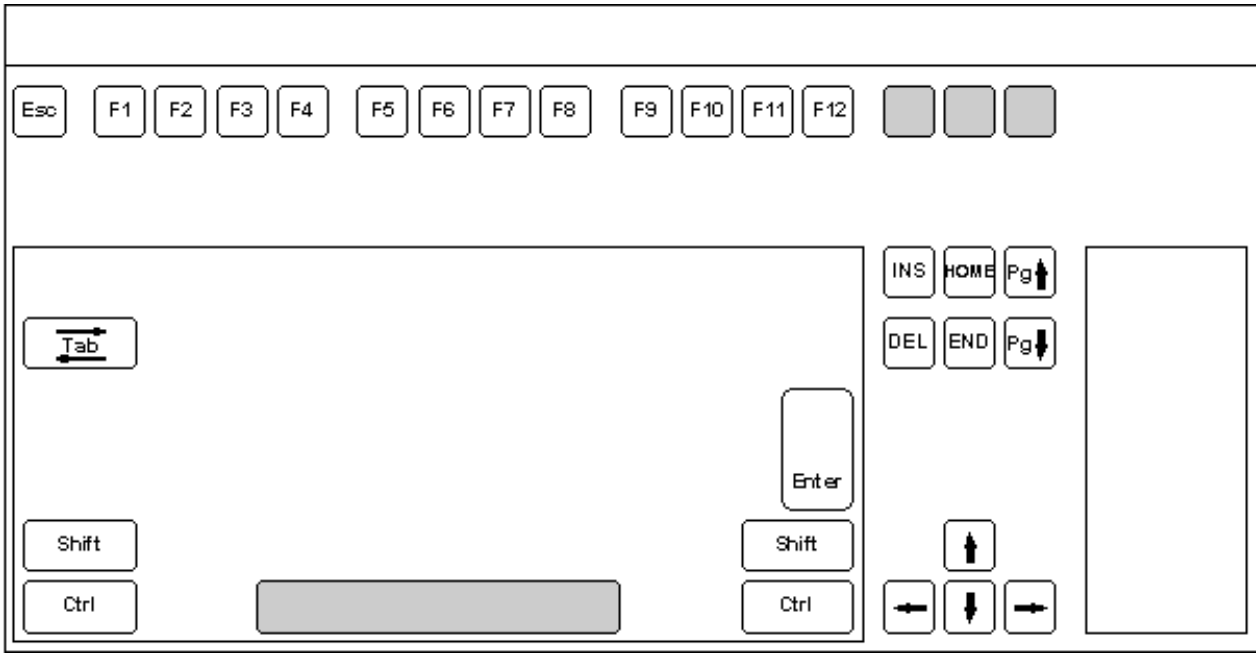
Keyboard Layout vt100



Keyboard Layout vt220



Keyboard Layout MF2



<ESC> <INS>: insert line or insert block (if MARK before)

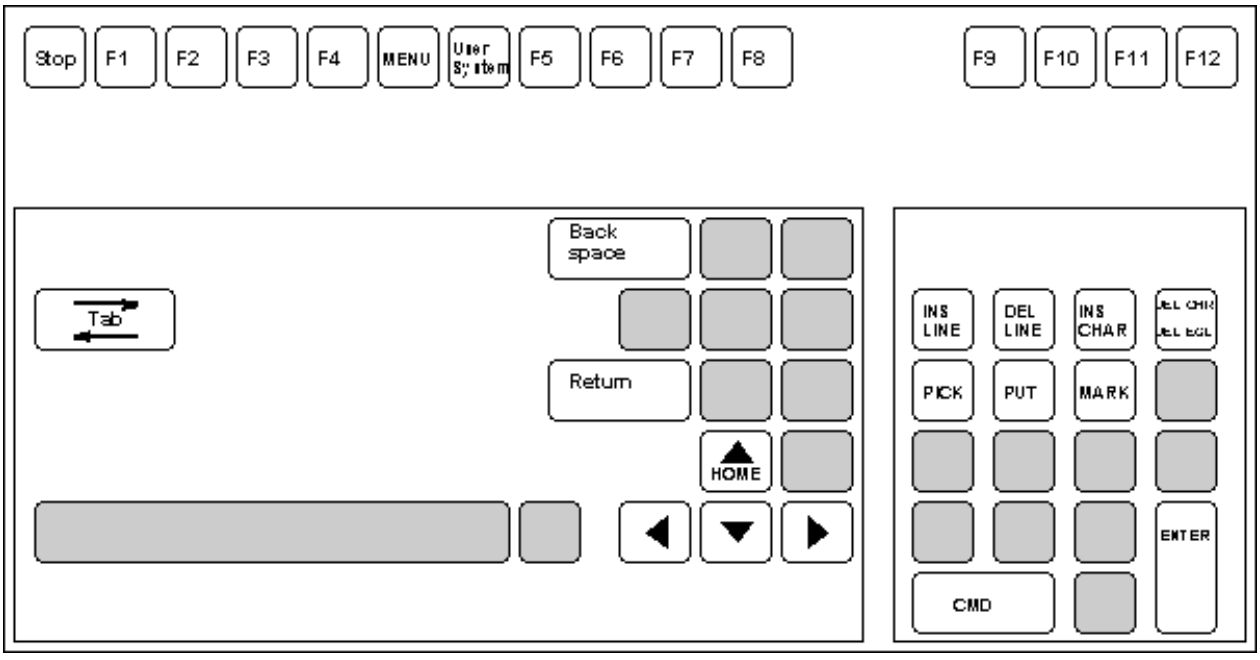
<ESC> <DEL>: delete line or delete block (if MARK before)

<CTRL> <F1>: CMD (jumping to command line) DBTERM=ansi

<CTRL> <I>: refresh screen DBTERM=ansi-gra

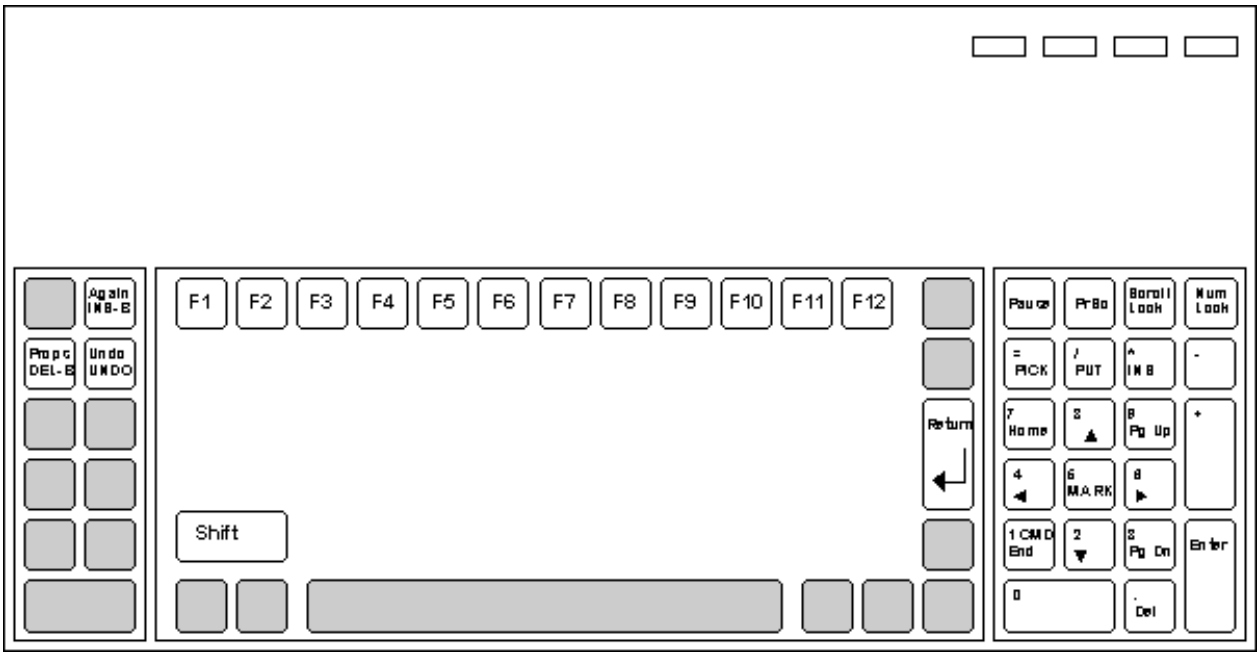
<CTRL> <x>: delete from cursor position to the end of line DBTERM=ansi-col

Keyboard Layout vt100_hp



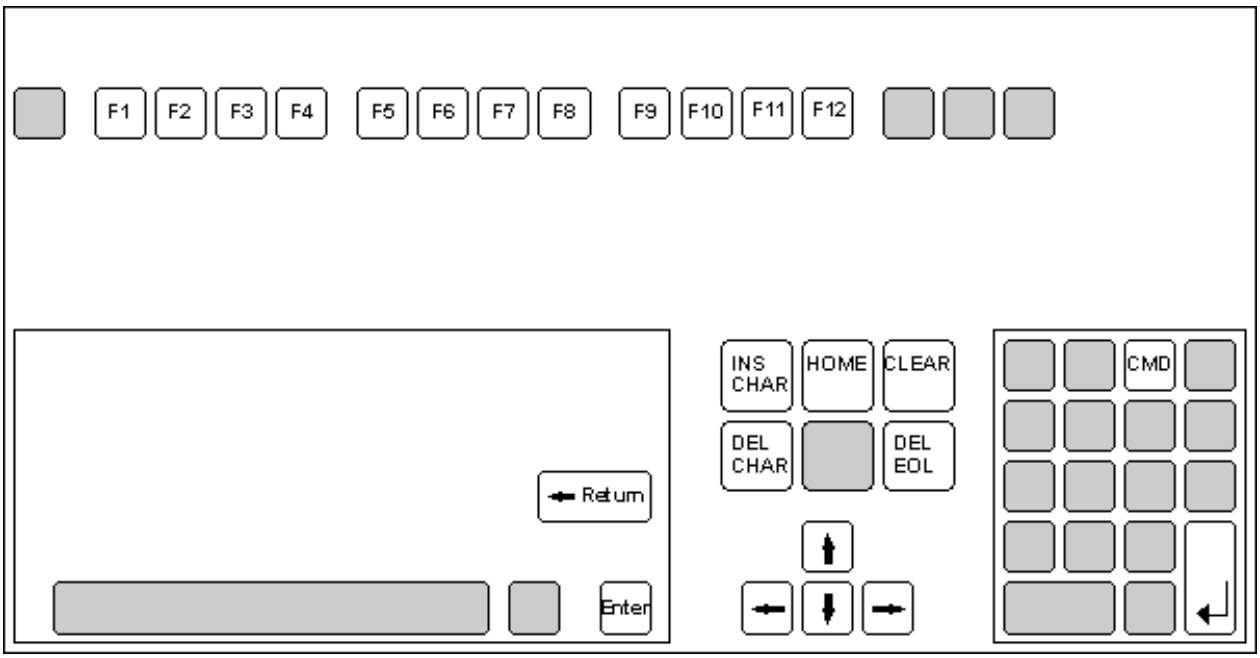
The second line is activated via <ESC>.

Keyboard Layout Open Windows (SUN)



Type4 Keyboard	DBTERM=sqldbsun
	DBHIF=sqldb-ow

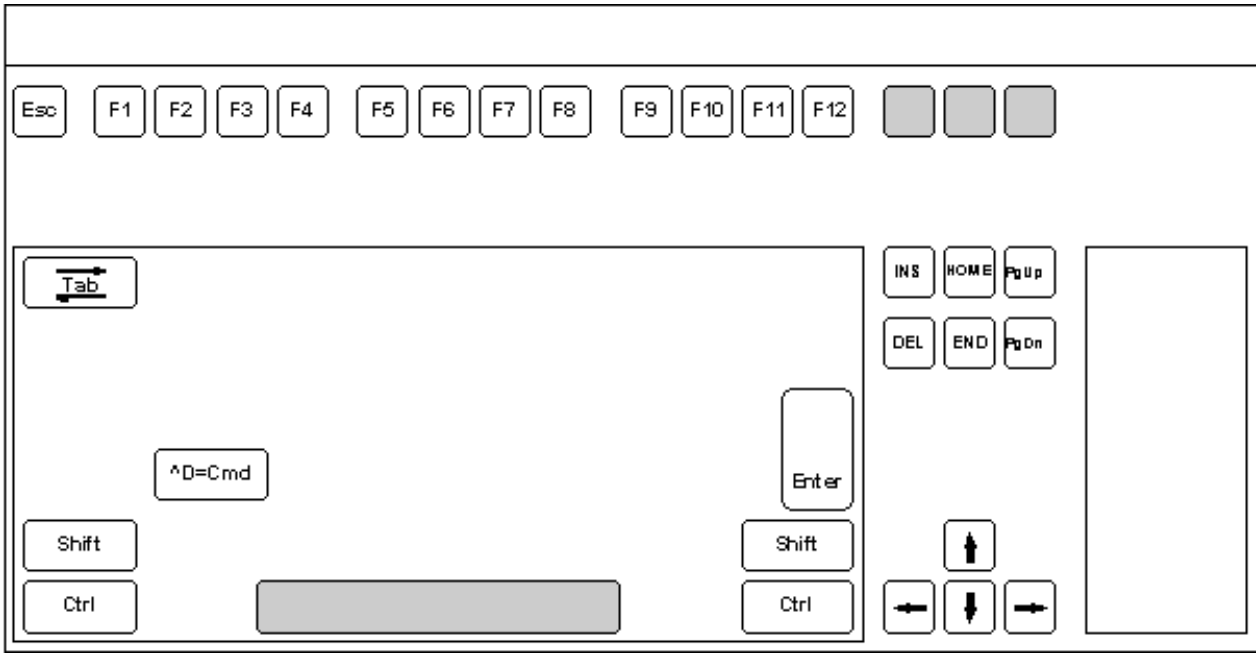
Keyboard Layout ibm3151



<CTRL> <INS CHAR>: <INS LINE> TERM=ibm3151

<CTRL> <DEL CHAR>: <DEL LINE> DBHIF=ibm3151

Keyboard Layout at386 (NCR)



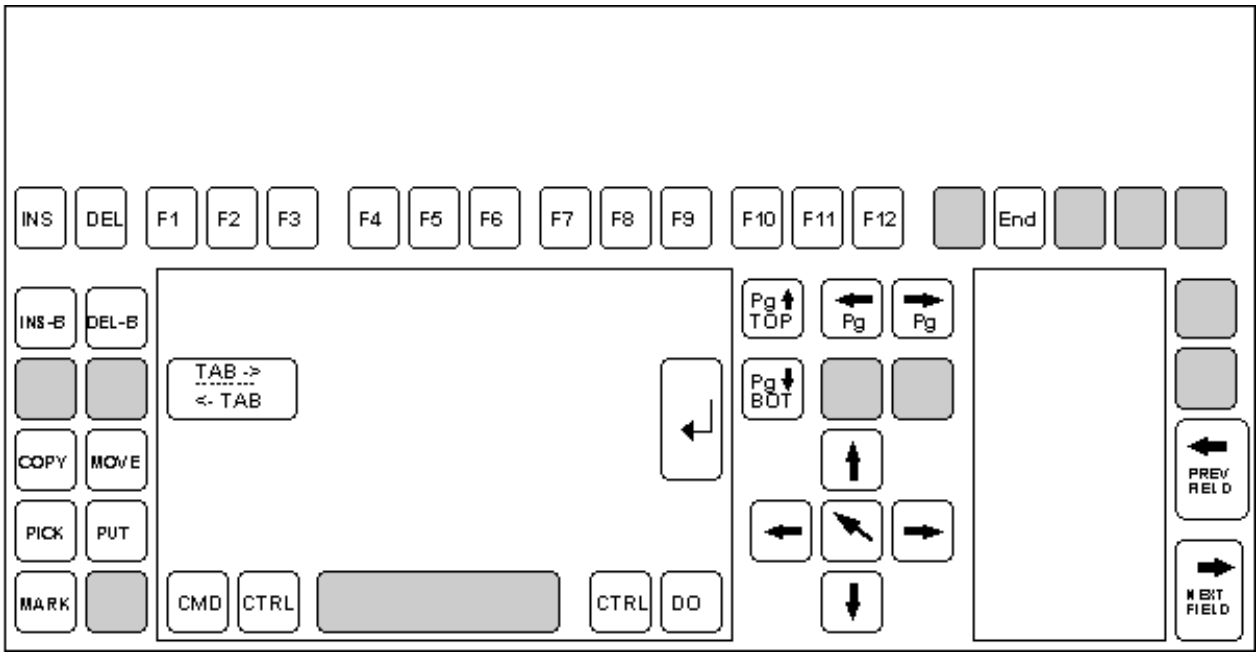
<ESC> <INS>: <Ins Line> TERM=at386

<ESC> <DEL>: <Del Line>

<ESC> <PgUp>: <Top>

<ESC> <PgDn>: <Bottom>

Keyboard Layout dap4x



The second line is activated using <CTRL>. DBHIF=dap4x